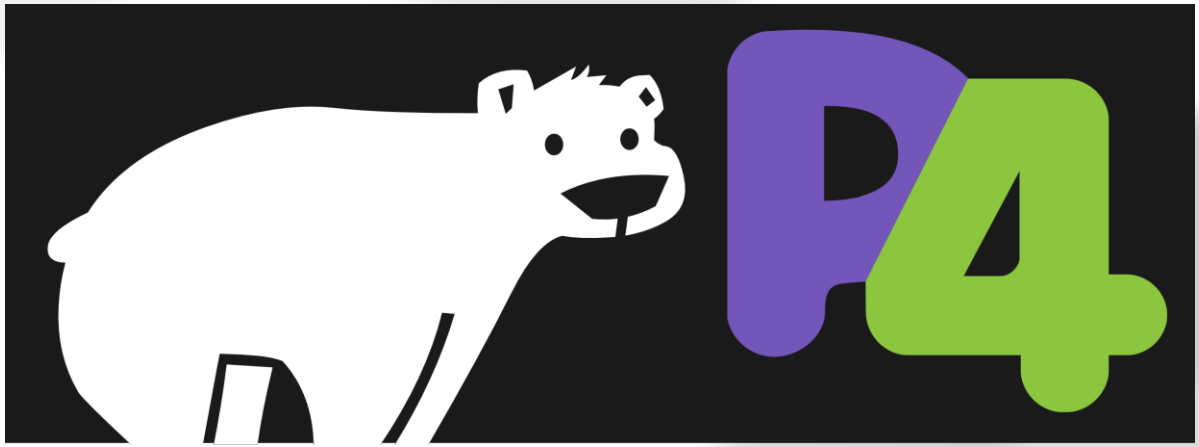


P4 Tutorial, Hot Chips 2017

Agenda

- **Background on Software Defined Networking**
- **P4: the data plane programming language**
- **Overview of the P4 toolchain**
- **P4 hardware implementations**
  - Tofino (Barefoot Networks)
  - FPGA (Xilinx)
  - Network Flow Processor / Agilio™ SmartNIC (Netronome)
- **Future directions**

Presenters

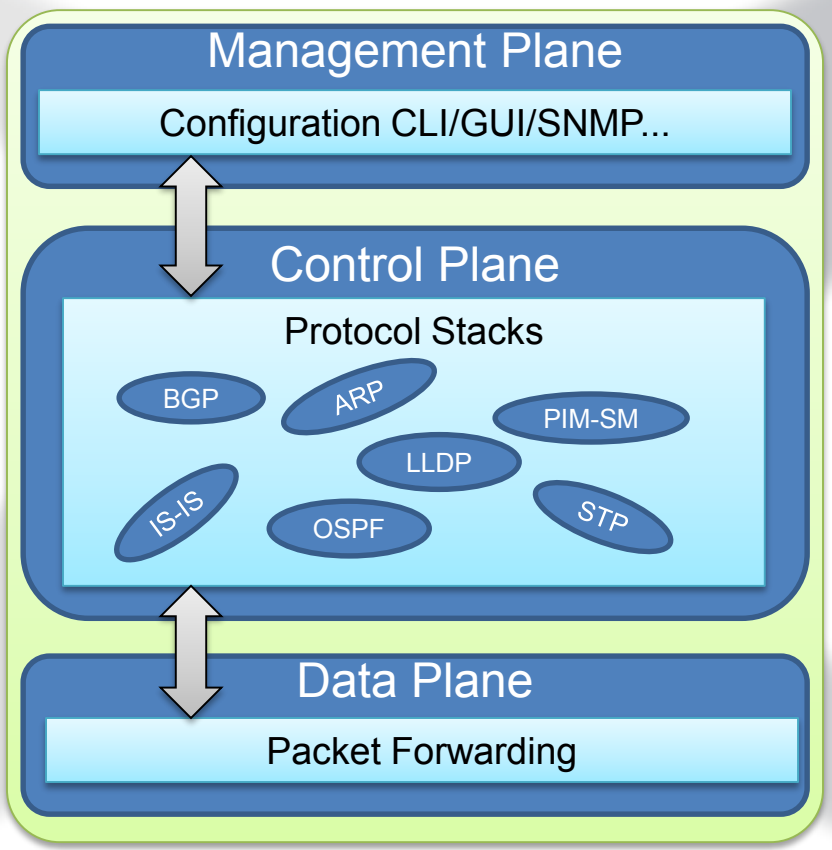
- Andy Fingerhut, Cisco
- Robert Halstead, Xilinx
- Jeongkeun “JK” Lee, Barefoot Networks
- Johann Tönsing, Netronome

Software Defined Networks

- Planes: data plane, control plane, management plane
- Software Defined Networking: logically centralized control
- Programming datapaths: dataplane protocol independence, flexibly specifying behaviors

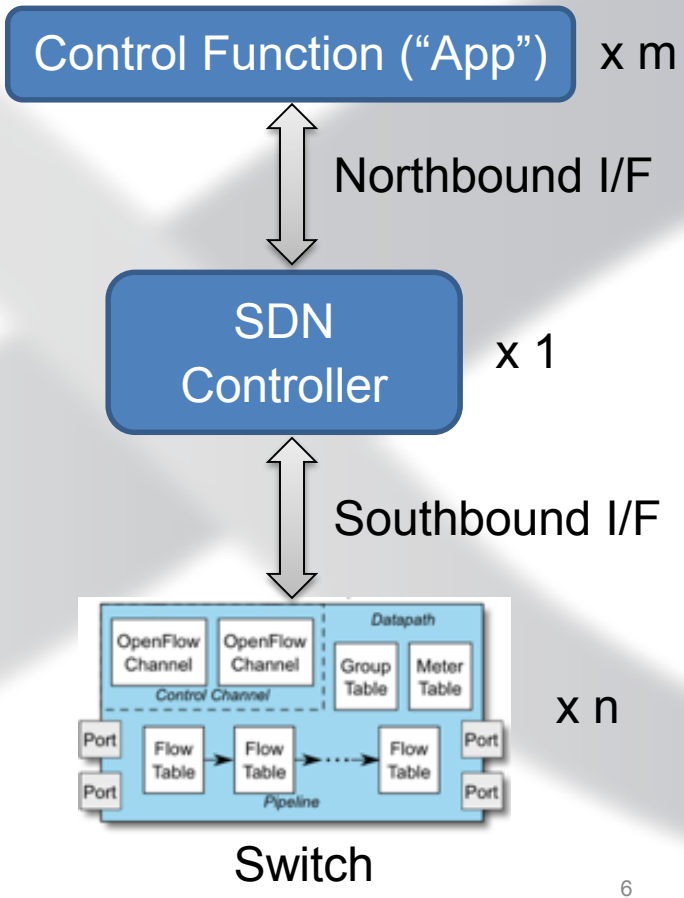
Standard Telecommunications Architecture

- **Traditional architecture consists of the three planes**
- **A Plane is a group of algorithms**
- **These algorithms**
  - Process different kinds of traffic
  - Have different performance requirements
  - Are designed using different methodologies
  - Are implemented using different programming languages
  - Run on different hardware



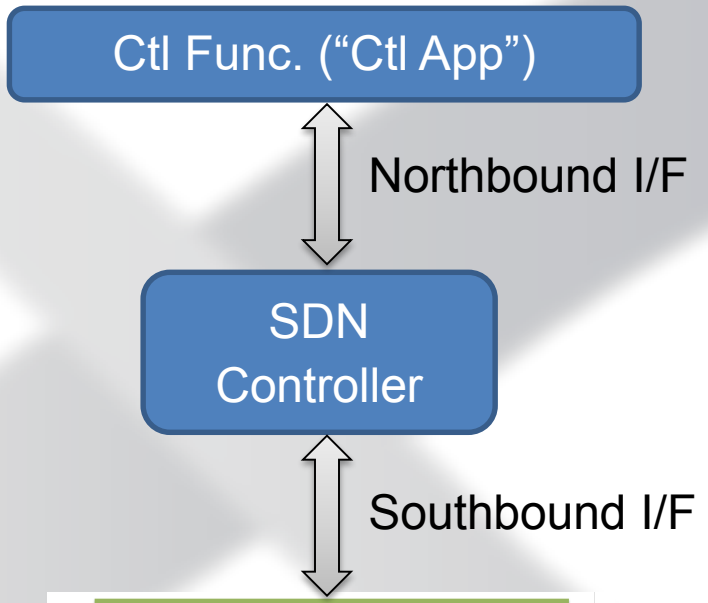
Software Defined Networking: Logically Centralized Control

- **Main contributions**
  - OpenFlow = standardized *model* (“architecture”) defining switch behavior (match / action)
  - OpenFlow = standardized *protocol* to interact with switch (download flow table entries, query statistics etc.)
  - *Concept* of *logically* centralized control via a single entity (“SDN controller”)
    - Simplifies control plane – e.g. compute optimal paths at one location (controller), vs. waiting for distributed routing algorithms to converge
- **Issues**
  - Limited interoperability between vendors => southbound I/F differences handled at controller (OpenFlow / netconf / JSON / XML variants)
  - Dataplane protocol evolution requires changes to standards (protocol and behavior)



SDN Evolved: (Dataplane) Protocol Independence

- **Main contributions**
  - Programs explicitly define dataplane protocols (parse tree)
  - Programs explicitly specify datapath behavior (matching, actions, queueing, global state…)
- **Issues**
  - Enabling SDN controller (or its equivalent, e.g. OpenStack) to take advantage of new protocols / behaviors: more flexible NBI+SBI
  - Introduces new entities: DP program, DP programmer => new lifecycle issues arise



Datapath_App.P4

Compile+ download

P4 Language

Brief History and Trivia

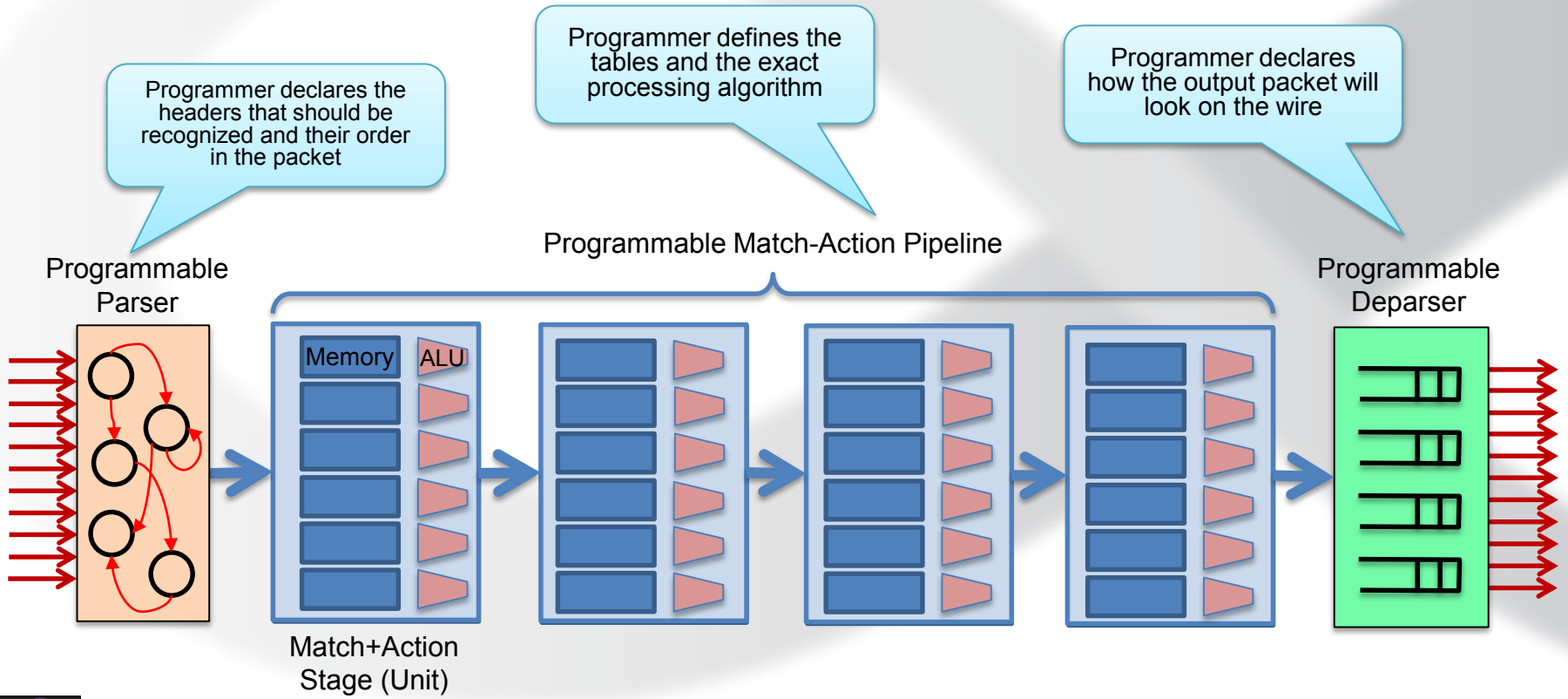
- May 2013: Initial idea and the name "P4"
- July 2014: First paper (SIGCOMM ACR)
- Aug 2014: First $P4_{14}$ Draft Specification (v0.9.8)
- Sep 2014: $P4_{14}$ Specification released (v1.0.0)
- Jan 2015: $P4_{14}$ v1.0.1
- Mar 2015: $P4_{14}$ v1.0.2
- Nov 2016: $P4_{14}$ v1.0.3
- May 2017: $P4_{14}$ v1.0.4

- Apr 2016: $P4_{16}$ – first commits
- Dec 2016: First $P4_{16}$ Draft Specification
- May 2017: $P4_{16}$ Specification released
- . . .

- Official Spelling `P4_16` on terminals, $P4_{16}$ in publications

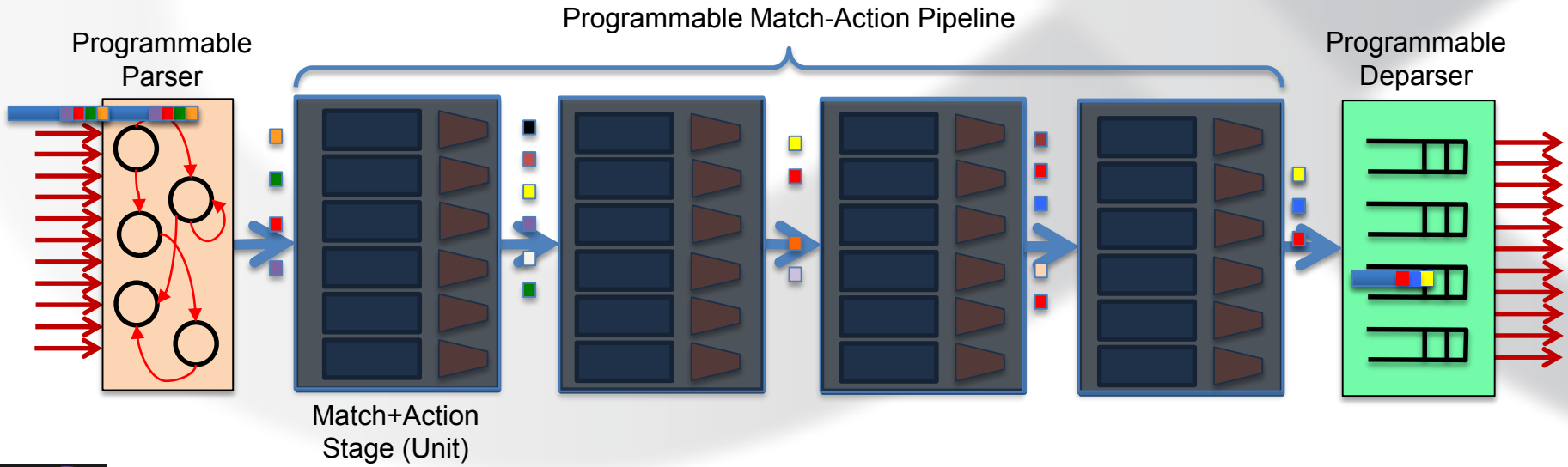
$P4_{16}$ Data Plane Model

PISA: Protocol-Independent Switch Architecture

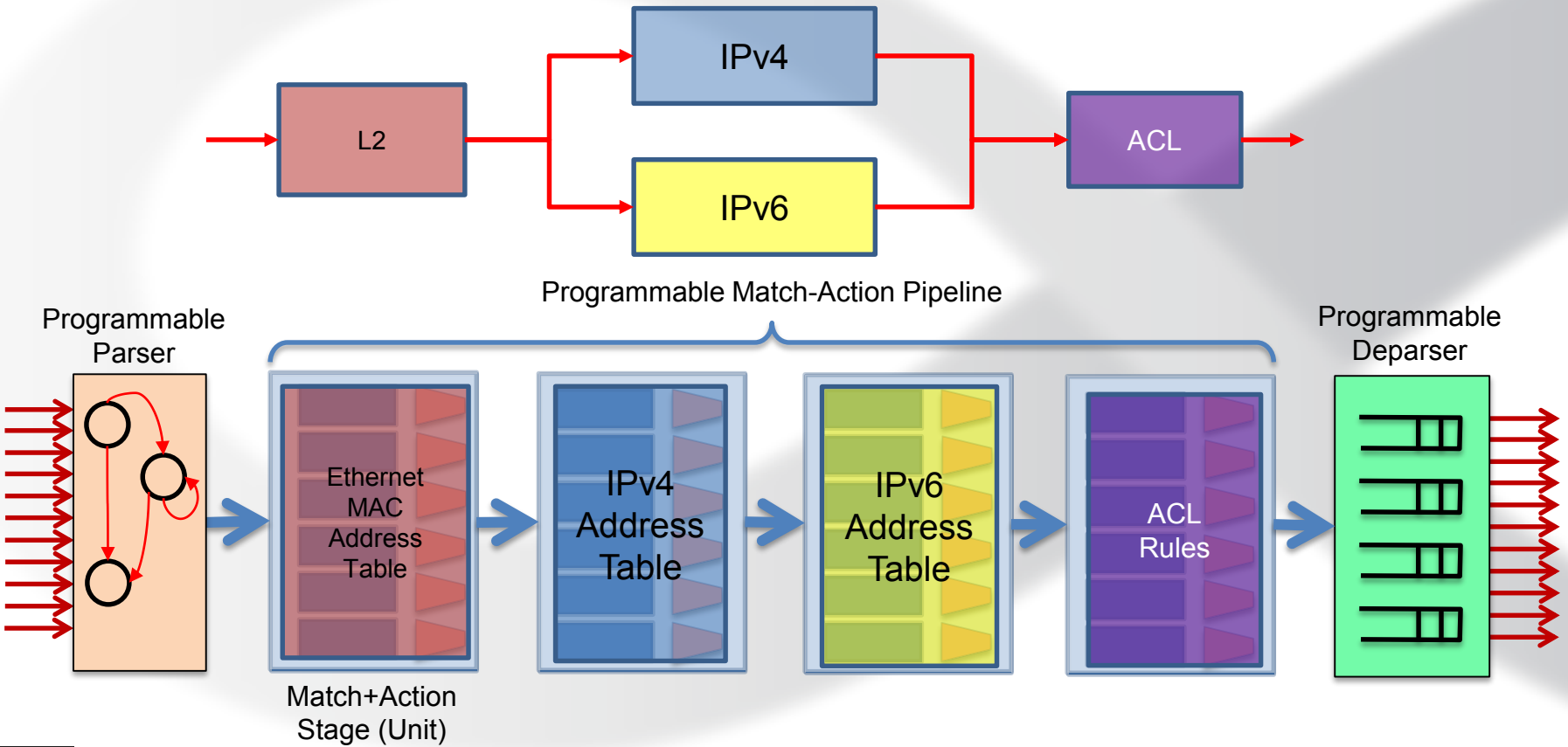


PISA in Action

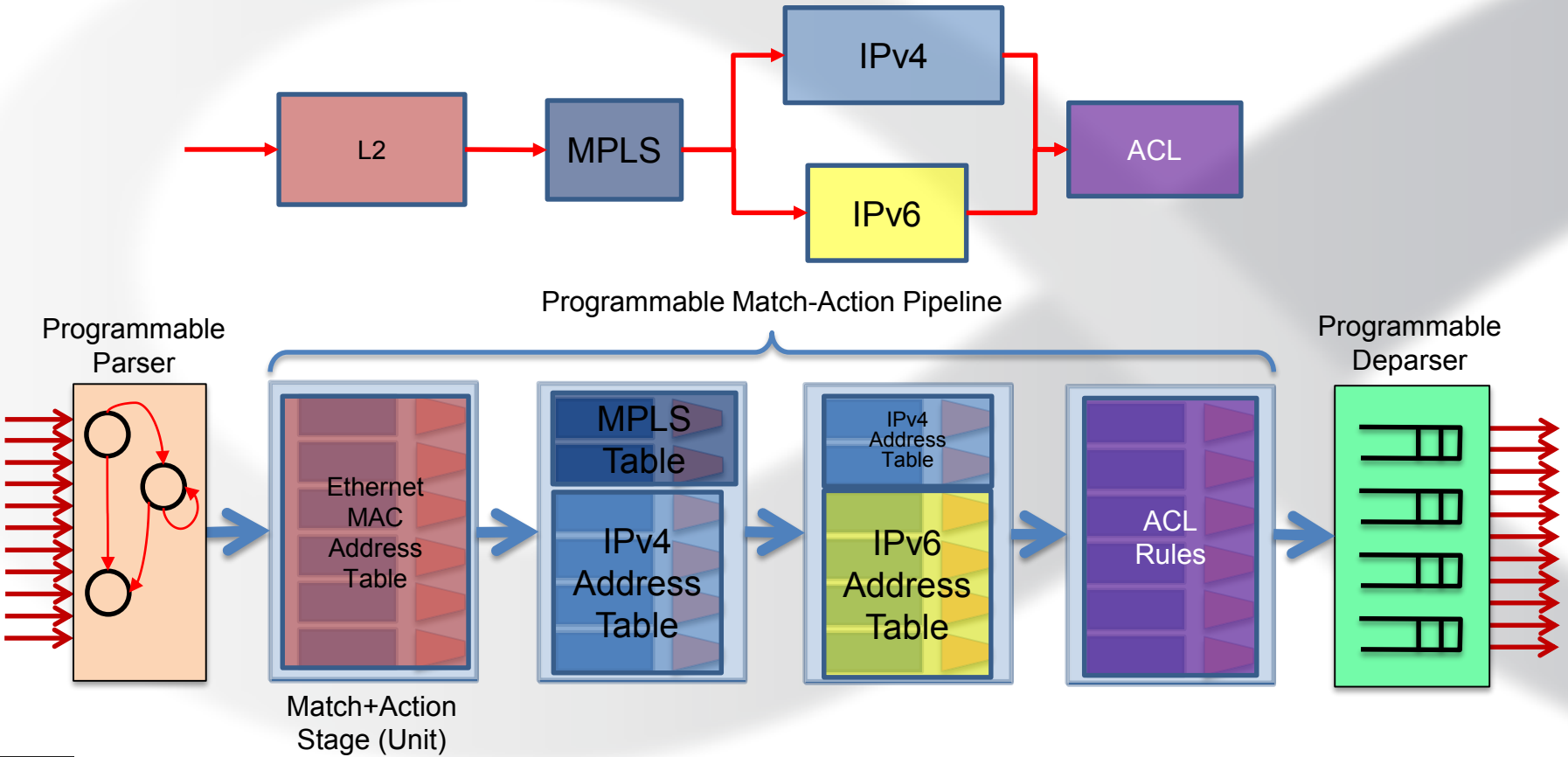
- Packet is parsed into individual headers (parsed representation)
- Headers and intermediate results can be used for matching and actions
- Headers can be modified, added or removed
- Packet is deparsed (serialized)



Mapping a Simple L3 Data Plane Program on PISA

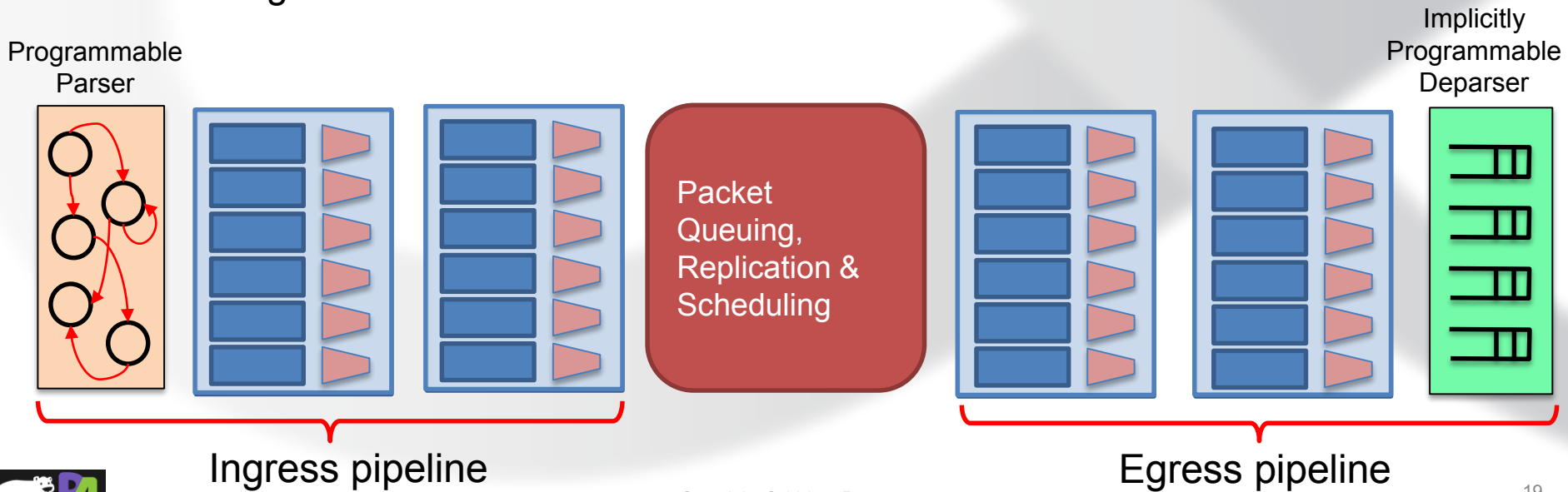


Mapping a More Complex Data Plane Program on PISA



$P4_{14}$ Switch Model

- **Ingress Pipeline**
- **Egress Pipeline**
- **Traffic Manager**
  - N:1 Relationships: Queueing, Congestion Control
  - 1:N Relationships: Replication
  - Scheduling



P4$_{16}$ Approach

Term	Explanation
P4 Target	An embodiment of a specific hardware implementation
P4 Architecture	A specific set of P4-programmable components, externs, fixed components and their interfaces available to the P4 programmer
P4 Platform	P4 Architecture implemented on a given P4 Target

$P4_{16}$ Basics

P4$_{16}$ Language Elements

Parsers

State machine, bitfield extraction

Controls

Tables, Actions, control flow statements

Expressions

Basic operations and operators

Data Types

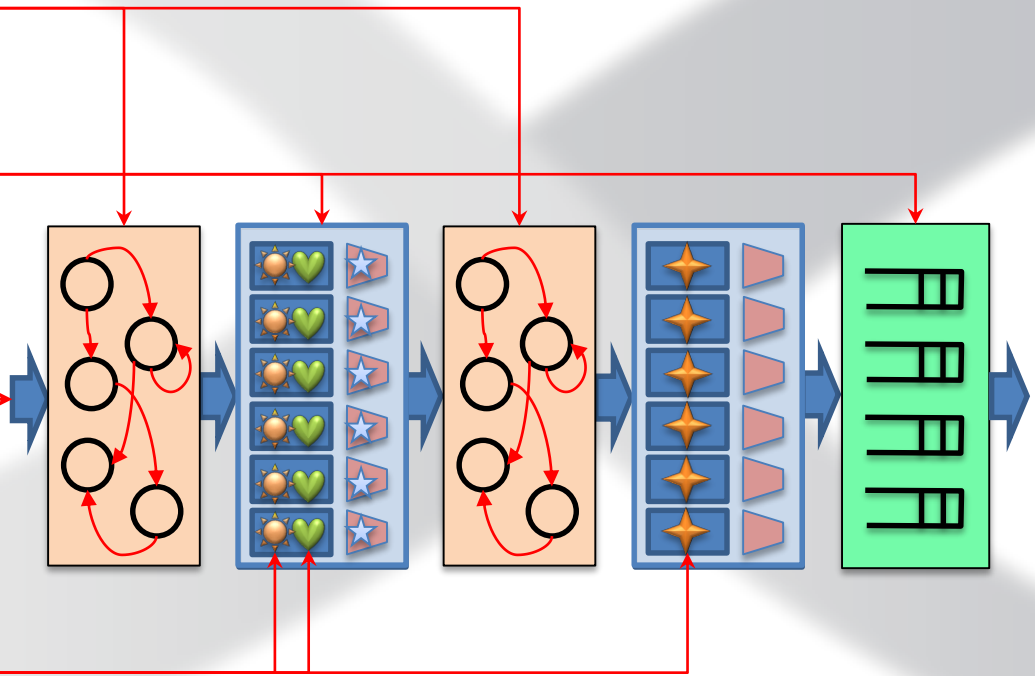
Bistrings, headers, structures, arrays

Architecture Description

Programmable blocks and their interfaces

Extern Libraries

Support for specialized components



$P4_{16}$ Data Types

- **Basic Data Types**
- **Derived Data Types**
  - Headers and metadata

Simple Header Definitions

Example: Declaring L2 headers

```
header ethernet_t {
    bit<48>    dstAddr;
    bit<48>    srcAddr;
    bit<16>    etherType;
}

header vlan_tag_t {
    bit<3>     pri;
    bit<1>     cfi;
    bit<12>    vid;
    bit<16>    etherType;
}

struct my_headers_t {
    ethernet_t    ethernet;
    vlan_tag_t[2] vlan_tag;
}
```

- **Basic Types**
  - **bit<n>** – Unsigned integer (bitsrting) of length n
    - **bit** is the same as **bit<1>**
  - **int<n>** – Signed integer of length n (>=2)
  - **varbit<n>** – Variable-length bitstring
- **Derived Types**
  - **header** – Ordered collection of members
    - Byte-aligned
    - Can be valid or invalid
    - Can contain bit<n>, int<n> and varbit<n>
  - **struct** – Unordered collection of members
    - No alignment restrictions
    - Can contain any basic or derived types
  - Header Stacks -- arrays of headers

Typedef

Example: Declaring a type for MAC address

```
typedef bit<48> mac_addr_t;

header ethernet_t {
    mac_addr_t  dstAddr;
    mac_addr_t  srcAddr;
    bit<16>     etherType;
}
```

- **Basic Types**
  - **bit<n>** – Unsigned integer (bitsrting) of length n
    - **bit** is the same as **bit<1>**
  - **int<n>** – Signed integer of length n (>=2)
  - **varbit<n>** – Variable-length bitstring
- **Derived Types**
  - **header** – Ordered collection of members
    - Byte-aligned
    - Can be valid or invalid
    - Can contain bit<n>, int<n> and varbit<n>
  - **struct** – Unordered collection of members
    - No alignment restrictions
    - Can contain any basic or derived types
  - Header Stacks -- arrays of headers
  - **typedef** – An alternative name for a type

Varbit

Example: Declaring IPv4 header

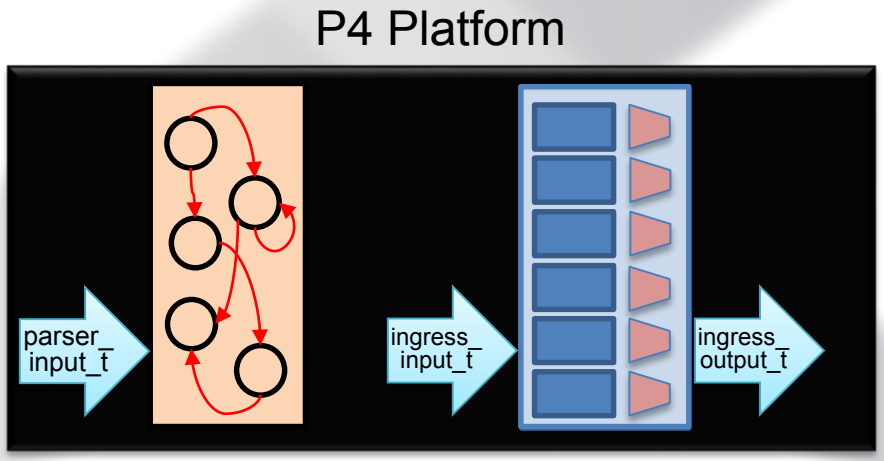
```
typedef bit<32> ipv4_addr_t;

header ipv4_t {
    bit<4>      version;
    bit<4>      ihl;
    bit<8>      diffserv;
    bit<16>     totalLen;
    bit<16>     identification;
    bit<3>      flags;
    bit<13>     fragOffset;
    bit<8>      ttl;
    bit<8>      protocol;
    bit<16>     hdrChecksum;
    ipv4_addr_t srcAddr;
    ipv4_addr_t dstAddr;
}

header ipv4_options_t {
    varbit<320> options;
}
```

- **Basic Types**
  - **bit<n>** – Unsigned integer (bitsrting) of length n
    - **bit** is the same as **bit<1>**
  - **int<n>** – Signed integer of length n (>=2)
  - **varbit<n>** – Variable-length bitstring
- **Derived Types**
  - **header** – Ordered collection of members
    - Byte-aligned
    - Can be valid or invalid
    - Can contain bit<n>, int<n> and varbit<n>
  - **struct** – Unordered collection of members
    - No alignment restrictions
    - Can contain any derived types
  - Header Stacks -- arrays of headers
  - **typedef** – An alternative name for a type

Using structs for Intrinsic Metadata

```
typedef bit<9>  port_id_t;  /* Switch port     */
typedef bit<16> mgid_t;     /* Multicast Group */
typedef bit<5>  qid_t;      /* Queue  ID       */

struct parser_input_t {
    port_id_t ingress_port;
    bit<1>    resubmit_flag;
}

struct ingress_input_t {
    portid_t      ingress_port;
    timestamp_t   ingress_timestamp;
}

struct ingress_output_t {
    portid_t      egress_port;
    mgid_t        mcast_group;
    bit<1>        drop_flag;
    qid_t         egress_queue;
}
```

- **Intrinsic Metadata is the data that a P4-programmable components can use to interface with the rest of the system**
- **These definitions come from the files, supplied by the vendor**



Declaring and Initializing Variables

```
bit<16>      my_var;
bit<8>       another_var = 5;

const bit<16> ETHERTYPE_IPV4 = 0x0800;
const bit<16> ETHERTYPE_IPV6 = 0x86DD;

ethernet_t   eth;
vlan_tag_t   vtag = { 3w2, 0, 12w13, 16w0x8847 };
```

Better than #define!

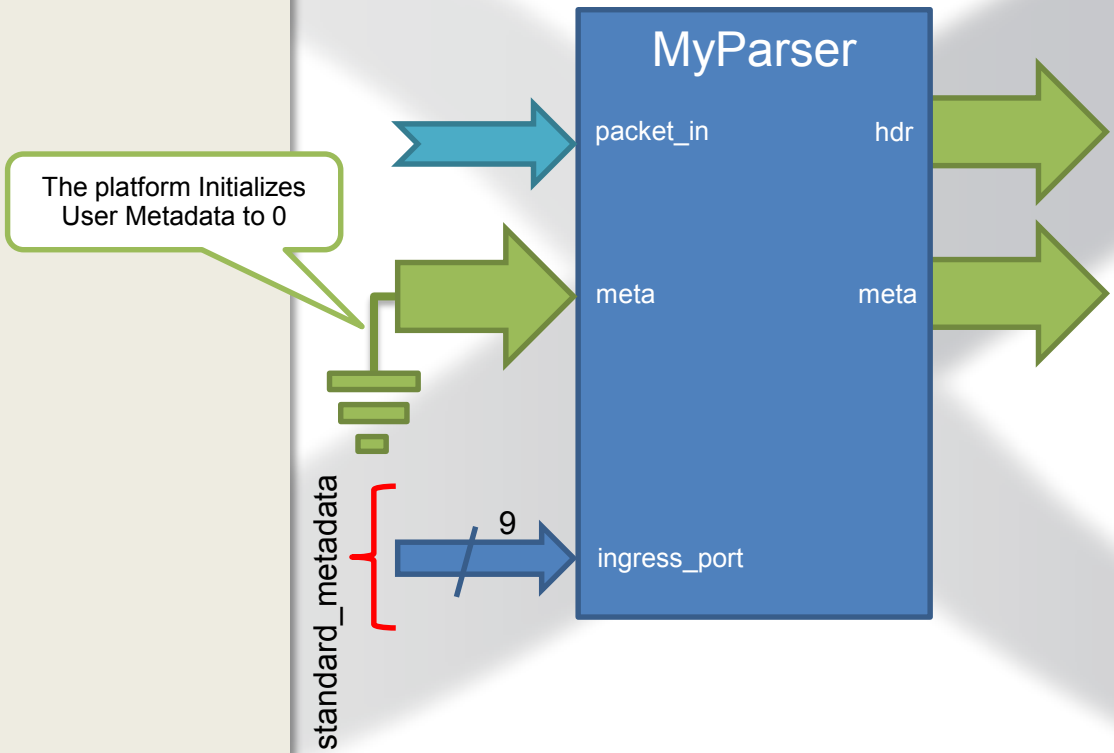
Safe constants with explicit widths

- **In $P4_{16}$ you can instantiate variables of both base and derived types**
- **Variables can be initialized**
  - Including the composite types
- **Constant declarations make for safer code**
- **Infinite width and explicit width constants**

Programming the Parser

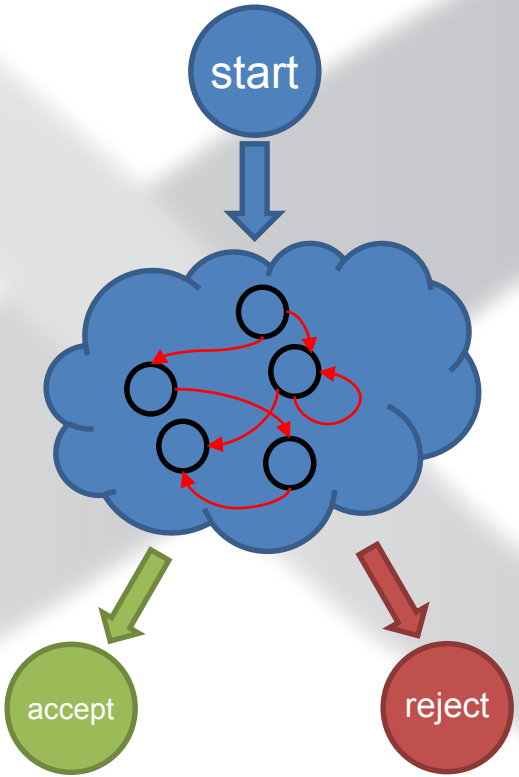
Parser Model (V1 Architecture)

```
#include <core.p4>
#include <v1model.p4>

/* User-defined inputs and outputs */
struct my_headers_t {
    ethernet_t    ethernet;
    vlan_tag_t[2] vlan_tag;
    ipv4_t        ipv4;
    ipv6_t        ipv6;
}

struct my_metadata_t {
    /* Nothing yet */
}

/* System-provided inputs. Others to follow */
struct standard_metadata_t {
    bit<9>   ingress_port;
    . . .
}

/* Parser Declaration */
parser MyParser(packet_in              packet,
                out   my_headers_t     hdr,
                inout my_metadata_t    meta,
                inout standard_metada_t standard_metadata)
{
  . . .
```



Parsers in $P4_{16}$

- **Parsers are special functions written in a state machine style**
- **Parsers have three predefined states**
  - start
  - accept
  - reject
    - Can be reached explicitly or implicitly
    - What happens in reject state is defined by an architecture
- **Other states are user-defined**



Implementing Parser State Machine

```
parser MyParser(packet_in               packet,
                out   my_headers_t      hdr,
                inout my_metadata_t     meta,
                in    standard_metadata_t  standard_metadata)
{
    state start {
        packet.extract(hdr.ethernet);
        transition select(hdr.ethernet.etherType) {
            0x8100 &&& 0xEFFF : parse_vlan_tag;
            0x0800  : parse_ipv4;
            0x86DD  : parse_ipv6;
            0x0806  : parse_arp;
            default : accept;
        }
    }

    state parse_vlan_tag {
        packet.extract(hdr.vlan_tag.next);
        transition select(hdr.vlan_tag.last.etherType) {
            0x8100 : parse_vlan_tag;
            0x0800 : parse_ipv4;
            0x86DD : parse_ipv6;
            0x0806 : parse_arp;
            default : accept;
        }
    }
```

```
    state parse_ipv4 {
        packet.extract(hdr.ipv4);
        transition select(hdr.ipv4.ihl) {
            0 .. 4: reject;
                5: accept;
          default: parse_ipv4_options;
    }

    state parse_ipv4_options {
        packet.extract(hdr.ipv4.options,
                       (hdr.ipv4.ihl - 5) << 2);
        transition accept;
    }

    state parse_ipv6 {
        packet.extract(hdr.ipv6);
        transition accept;
    }
```

Lookahead

Example: Typical MPLS Heuristic

```
header ip46_t {    /* Common for both IPv4 and IPv6 */
    bit<4> version;
    bit<4> reserved;
}

state parse_mpls {
    packet.extract(hdr.mpls.next);
    transition select(hdr.mpls.last.bos) {
        0: parse_mpls;
        1: guess_mpls_payload;
    }
}

state guess_mpls_payload {
    transition select(packet.lookahead<ip46_t>().version) {
            4 : parse_inner_ipv4;
            6 : parse_inner_ipv6;
        default : parse_inner_ethernet;
    }
}
```

- **lookahead is a generic method**
  - Compiler generates the method with the proper return type (ipv4_t) at the compile time
- **Returns the packet data without advancing the cursor**
  - The packet length is still checked
- **Much safer and easier to use than bit offsets**

Programming the Match-Action Pipeline

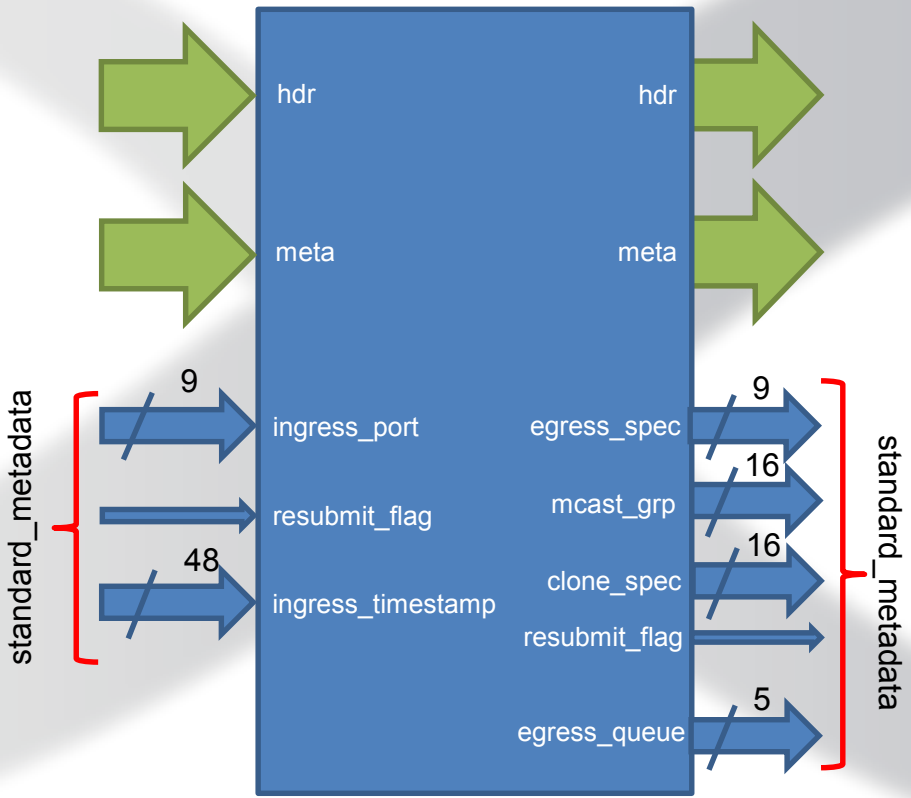
- Controls
- Actions
- Tables

Controls in P4

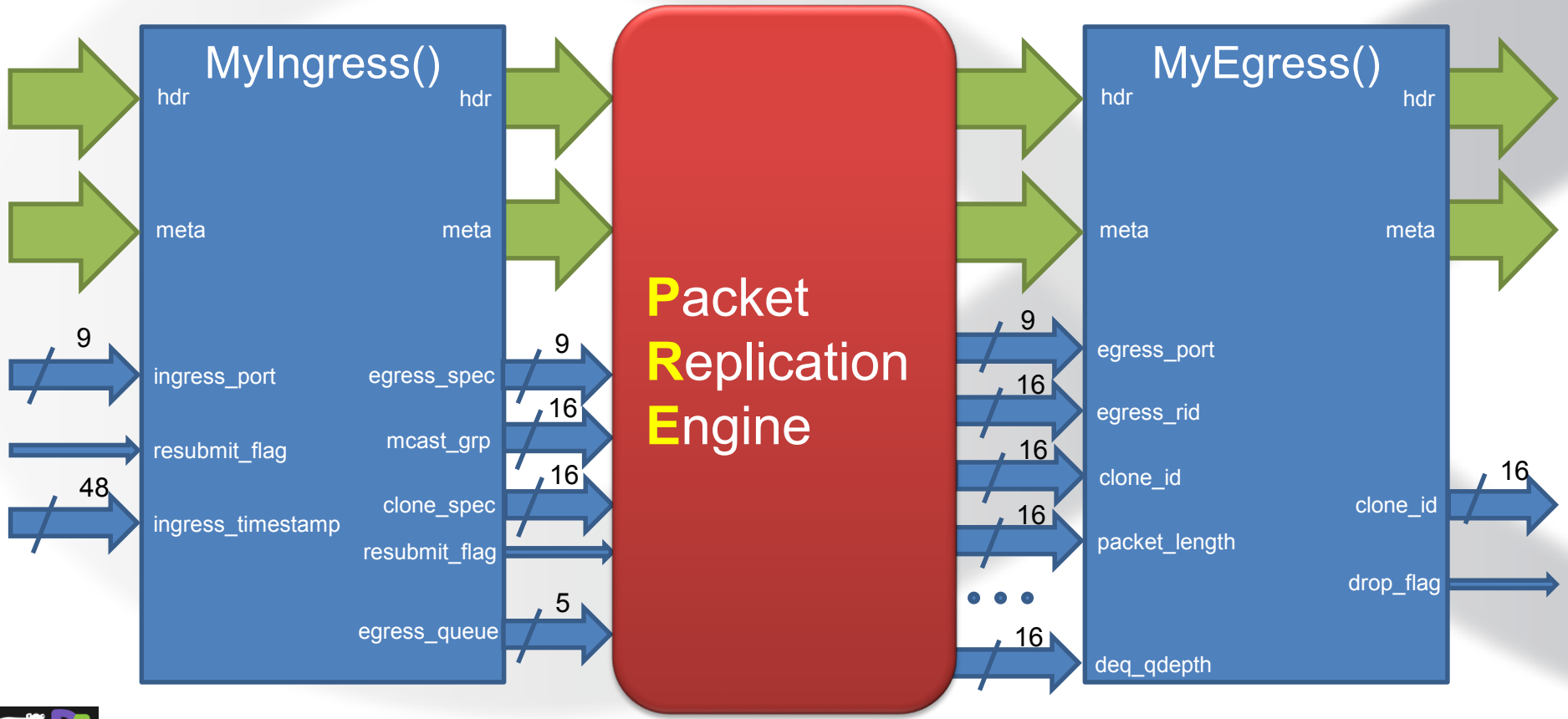
- **Very similar to C functions without loops**
  - Algorithms should be representable as Direct Acyclic Graphs (DAG)
- **Represent all kinds of processing that are expressible as DAG:**
  - Match-Action Pipelines
  - Deparsers
  - Additional processing (checksum updates)
- **Interface with other blocks via user- and architecture-defined data**

Simple Reflector (V1 Architecture)

```
control MyIngress(
    inout my_headers_t        hdr,
    inout my_metadata_t       meta,
    inout standard_metadata_t  standard_metadata)
{
    bit<48> tmp;

    apply {
        tmp = hdr.ethernet.dstAddr;
        hdr.ethernet.dstAddr = hdr.ethernet.srcAddr;
        hdr.ethernet.srcAddr = tmp;

        standard_metadata.egress_spec =
                        standard_metadata.ingress_port;
    }
}
```



How does it work? (V1 Architecture)



Simple Actions and Expressions

```
const bit<9> DROP_PORT = 511; /* Specific to V1 architecture */

action mark_to_drop() {                  /* Already defined in v1model.p4 */
   standard_metadata.egress_spec = DROP_PORT;
   standard_metadata.mgast_grp   = 0;
}

control MyIngress(inout my_headers_t        hdr,
                  inout my_metadata_t       meta,
                  inout standard_metadata_t  standard_metadata)
{
   /* Local Declarations */

   action swap_mac(inout bit<48> dst, inout bit<48> src) {
      bit<48> tmp;
      tmp = dst; dst = src; src = tmp;
   }

   action reflect_to_other_port() {
      standard_metadata.egress_spec =
                     standard_metadata.ingress_port ^ 1;
   }
```

- **Very similar to C functions**
- **Can be declared inside a control or globally**
- **Parameters have type and direction**
- **Variables can be instantiated inside**

- **Standard Arithmetic and Logical operations are supported**
  - +, -, *
  - ~, &, |, ^, >>, <<
  - ==, !=, >, >=, <, <=
  - No division/modulo
- **Additional operations:**
  - Bit-slicing: [m:l]
    - Works as l-value too
  - Bit Concatenation: ++

Simple Actions and Expressions

```
const bit<9> DROP_PORT = 511;  /* V1 Architecure-specific */

action mark_to_drop() {  /* Already defined in v1model.p4 */
   standard_metadata.egress_spec = DROP_PORT;
   standard_metadata.mgast_grp   = 0;
}

control MyIngress(inout my_headers_t       hdr,
                  inout my_metadata_t      meta,
                  inout standard_metadata_t standard_metadata)
{
   /* Local Declarations */

   action swap_mac(inout bit<48> dst, inout bit<48> src) {
      bit<48> tmp;
      tmp = dst; dst = src; src = tmp;
   }

   action reflect_to_other_port() {
      standard_metadata.egress_spec =
                     standard_metadata.ingress_port ^ 1;
   }
```

```
   /* The body of the control */

   apply {
      if (hdr.ethernet.dstAddr[40:40] == 0x1) {
         mark_to_drop();
      } else {
         swap_mac(hdr.ethernet.dstAddr,
                  hdr.ethernet.srcAddr);
         reflect_to_other_port();
      }
   }
}
```

Actions Galore: Operating on Headers

Example: Encapsulating IPv4 into a 2-label MPLS packet

```
header mpls_t {
    bit<20> label;
    bit<3>  exp;
    bit<1>  bos;
    bit<8>  ttl;
}

action ipv4_in_mpls(in bit<20> label1, in bit<20> label2) {
    hdr.mpls[0].setValid();
    hdr.mpls[0].label = label1;
    hdr.mpls[0].exp   = 0;
    hdr.mpls[0].bos   = 0;
    hdr_mpls[0].ttl   = 64;

    hdr.mpls[1].setValid();
    hdr.mpls[1] = { label2, 0, 1, 128 };

    if (hdr.vlan_tag.isValid()) {
        hdr.vlan_tag.etherType = 0x8847;
    } else {
        hdr.ethernet.etherType = 0x8847;
    }
}
```

if() statements are allowed in actions too!

- **Header Validity bit manipulation:**
  - header.**setValid**() – add_header
  - header.**setInvalid**() – remove_header
  - header.**isValid**()
- **Header Assignment**
  - From tuples

Actions Galore: Bit Manipulation

Example: **Forming Ethernet MAC address for IPv4 Multicast Packets**

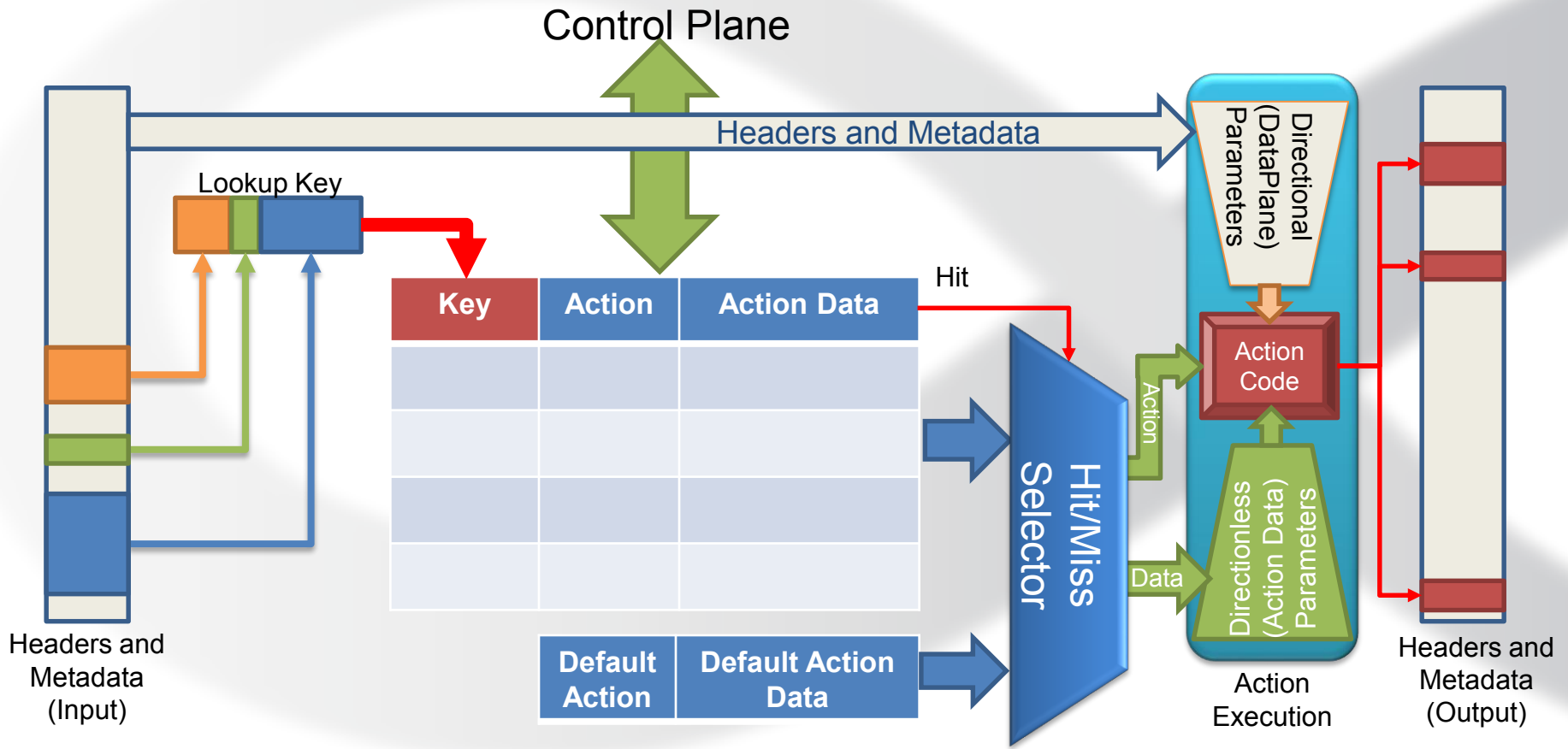
```
action set_ipmcv4_mac_da_1() {
   hdr.ethernet.dstAddr = 24w0x01005E ++ 1w0 ++
                          hdr.ipv4.dstAddr[22:0];
}

action set_ipmcv4_mac_da_2() {
   hdr.ethernet.dstAddr[47:24] = 0x01005E;
   hdr.ethernet.dstAddr[23:23] = 0;;
   hdr.ethernet.dstAddr[22:0]  = hdr.ipv4.dstAddr[22:0];
}
```

- **Special Operations for bit manipulation:**
  - Bit-string concatenation
  - Bit-slicing

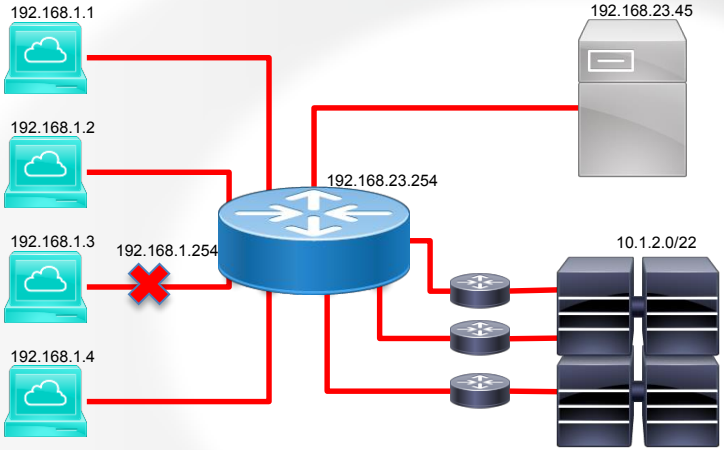
Match-Action Tables

- **The fundamental units of the Match-Action Pipeline**
  - What to match on and match type
  - A list of *possible* actions
  - Additional **properties**
    - Size
    - Default Action
    - Entries
    - etc.
- **Each table contains one or more entries (rows)**
- **An entry contains:**
  - A specific key to match on
  - A **single** action
    - to be executed when a packet matches the entry
  - (Optional) action data

Tables: Match-Action Processing



Example: Basic IPv4 Forwarding



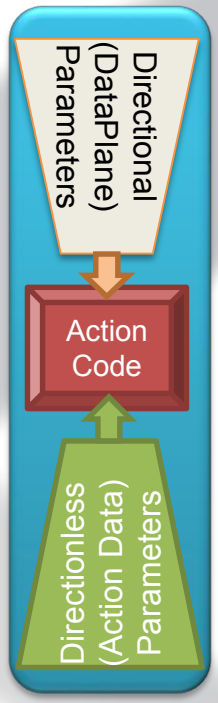
Key	Action	Action Data
192.168.1.1	l3_switch	port= mac_da= mac_sa= vlan=
192.168.1.2	l3_switch	port= mac_da= mac_sa= vlan=…
192.168.1.3	l3_drop	
192.168.1.254	l3_l2_switch	port=
192.168.1.0/24	l3_l2_switch	port=
10.1.2.0/22	l3_switch_ecmp	ecmp_group=

- **Data Plane (P4) Program**
  - Defines the format of the table
    - Key Fields
    - Actions
    - Action Data
  - Performs the lookup
  - Executes the chosen action
- **Control Plane (IP stack, Routing protocols)**
  - Populates table entries with specific information
    - Based on the configuration
    - Based on automatic discovery
    - Based on protocol calculations

Defining Actions for L3 forwarding

```
action l3_switch(bit<9>  port,
                 bit<48> new_mac_da,
                 bit<48> new_mac_sa,
                 bit<12> new_vlan)
{
   /* Forward the packet to the specified port */
   standard_metadata.metadata.egress_spec = port;

   /* L2 Modifications */
   hdr.ethernet.dstAddr   = new_mac_da;
   hdr.ethernet.srcAddr   = mac_sa;
   hdr.vlan_tag[0].vlanid = new_vlan;

   /* IP header modification (TTL decrement) */
   hdr.ipv4.ttl = hdr.ipv4.ttl – 1;
}

action l3_l2_switch(bit<9> port) {
   standard_metadata.metadata.egress_spec = port;
}

action l3_drop() {
   mark_to_drop();
}
```

- **Actions can use two types of parameters**
  - Directional (from the Data Plane)
  - Directionless (from the Control Plane)
- **Actions that are called directly:**
  - Only use directional parameters
- **Actions used in tables:**
  - Typically use direction-less parameters
  - May sometimes use directional parameters too



Action Execution

Match-Action Table (Exact Match)

Example: A typical L3 (IPv4) Host table

```
table ipv4_host {
   key = {
      meta.ingress_metadata.vrf   : exact;
      hdr.ipv4.dstAddr            : exact;
   }
   actions = {
      l3_switch;  l3_l2_switch;
      l3_drop;    noAction;
   }
   default_action = noAction();
   size = 65536;
}
```

These are the only possible actions. Each particular entry can have only ONE of them.

```
/* Defined in core.p4 */
action noAction() { }
```

vrf	ipv4.dstAddr	action	data
1	192.168.1.10	l3_switch	port_id= mac_da= mac_sa=
100	192.168.1.10	l3_l2_switch	port_id=<CPU>
1	192.168.1.3	l3_drop	
	DEFAULT	noAction	

Match-Action Table (Longest Prefix Match)

Example: A typical L3 (IPv4) Routing table

```
table ipv4_lpm {
    key = {
        meta.ingress_metadata.vrf   : exact;
        hdr.ipv4.dstAddr            : lpm;
    }
    actions = {
        l3_switch;
        l3_l2_switch;
        l3_switch_nexthop(meta.l3.nexthop_info);
        l3_switch_ecmp(meta.l3.nexthop_info);
        l3_drop;  noAction;
    }
    const default_action = l3_l2_switch(CPU_PORT);
    size = 16384;
}
```

Different fields can use different match types

Prefix length also serves as a priority indicator

vrf	ipv4.dstAddr / prefix	action	data
1	192.168.1.0 / 24	l3_l2_switch	port_id=3
10	10.0.16.0 / 22	l3_ecmp	ecmp_index=12
1	192.168.0.0 / 16	l3_switch_nexthop	nexthop_index=451
1	0.0.0.0 / 0	l3_switch_nexthop	nexthop_index=1
	DEFAULT	l3_l2_switch	port_id=CPU_PORT

Match-Action Table (Ternary Match)

Example: A more powerful L3 (IPv4) Routing table

```
table ipv4_lpm {
    key = {
        meta.ingress_metadata.vrf   : ternary;
        hdr.ipv4.dstAddr            : ternary;
    }
    actions = {
        l3_switch;
        l3_l2_switch;
        l3_switch_nexthop(meta.l3.nexthop_info);
        l3_switch_ecmp(meta.l3.nexthop_info);
        l3_drop;
        noAction;
    }
    const default_action = l3_l2_switch(CPU_PORT);
    size = 16384;
}
```

Explicitly Specified Priority

Prio	vrf / mask	ipv4.dstAddr / mask	action	data
100	0x001/0xFFF	192.168.1.5 / 255.255.255.255	l3_swith_nexthop	nexthop_index=10
10	0x000/0x000	192.168.2.0/255.255.255.0	l3_switch_ecmp	ecmp_index=25
10	0x000/0x000	192.168.3.0/255.255.255.0	l3_switch_nexthop	nexthop_index=31
5	0x000/0x000	0.0.0.0/0.0.0.0	l3_l2_switch	port_id=64

Using Tables in the Controls

```
control MyIngress(inout my_headers_t       hdr,
                  inout my_metadata_t      meta,
                  inout standard_metadata_t standard_metadata)
{
    /* Declarations */
    action l3_switch(...) { . . . }
    action l3_l2_switch(...) { . . . }
    . . .
    table assign_vrf { . . . }
    table ipv4_host  { . . . }
    table ipv6_host  { . . . }

    /* Code */
    apply {
        assign_vrf.apply();
        if (hdr.ipv4.isValid()) {
            ipv4_host.apply();
        }
    }
}
```

- **Declare Actions**
  - Declaration is instantiation
- **Declare Tables**
  - Declaration is instantiation
- **Apply() Tables – Perform Match-Action**
  - Make sure the table matches on valid headers

Using the Match Results

```
apply {
   . . .
   if (hdr.ipv4.isValid()) {
      if (!ipv4_host.apply().hit) {
         ipv4_lpm.apply();
      }
   }
}

apply {
   . . .
   switch (ipv4_lpm.apply().action_run) {
      l3_switch_nexthop: { nexthop.apply(); }
        l3_switch_ecmp: { ecmp.apply(); }
             l3_drop: { exit; }
             default: { /* Not needed. Do nothing */ }
   }
}
```

- **Apply method returns a special result:**
  - A boolean, representing the hit
  - An enum, representing a selected action
- **Switch() statement**
  - Only used for the results of match-action
  - Each case should be a block statement
  - Default case is optional
    - Means “any action” not “default action”
- **Exit and Return Statements**
  - return – go to the end of the current control
  - exit – go to the end of the top-level control
  - Useful to skip further processing

Match Kinds (types)

```
/* core.p4 */

match_kind {
    exact,
    ternary,
    lpm
}

/* v1model.p4 */

match_kind {  /* Augments the standard definition }
    range,
    selector
}

/* Some other architecture */

match_kind {
    regexp,
    range,
    fuzzy,
    telepathy
}
```

- **match_kind is a special type in P4**
- **core.p4 defines three basic match kinds**
  - Exact match
  - Ternary match
  - LPM match
- **Architectures can add their own match kinds**

Advanced Matching

```
table classify_ethernet {
    key = {
        hdr.ethernet.dstAddr      : ternary;
        hdr.ethernet.srcAddr      : ternary;
        hdr.vlan_tag[0].isValid() : ternary;
        hdr.vlan_tag[1].isValid() : ternary;
    }
    actions = {
        malformed_ethernet;
        unicast_untagged;
        unicast_single_tagged;
        unicast_double_tagged;
        multicast_untagged;
        multicast_single_tagged;
        multicast_double_tagged;
        broadcast_untagged;
        broadcast_single_tagged;
        broadcast_double_tagged;
    }
```

- **Tables keys can be arbitrary expressions**
- **Check header validity**
  - header.isValid()
- **Use only important bits**
  - header.field[msb:lsb]
    - hdr.ipv6.dstAddr[127:64] : lpm;
- **Fantasy is your limit:**
  - hdr.ethernet.srcAddr ^ hdr.ethernet.dstAddr

Table Initialization

```
  /* Continued from previous slide */
  const entries = {
      /* { dstAddr, srcAddr, vlan_tag[0].isValid(), vlan_tag[1].isValid() } : action([action_data]) */
    { 48w000000000000,                  _,            _, _ } : malformed_ethernet(ETHERNET_ZERO_DA);
    { _,                          48w000000000000, _, _ } : malformed_ethernet(ETHERNET_ZERO_SA);
    { _,           48w010000000000 &&& 48w010000000000, _, _ } : malformed_ethernet(ETHERNET_MCAST_SA);
    { 48wFFFFFFFFFFFF,                  _,            0, _ } : broadcast_untagged();
    { 48wFFFFFFFFFFFF,                  _,            1, 0 } : broadcast_single_tagged();
    { 48wFFFFFFFFFFFF,                  _,            1, 1 } : broadcast_double_tagged();
    { 48w010000000000 &&& 48w010000000000, _,            0, _ } : multicast_untagged();
    { 48w010000000000 &&& 48w010000000000, _,            1, 0 } : multicast_single_tagged();
    { 48w010000000000 &&& 48w010000000000, _,            1, 1 } : multicast_double_tagged();
    { _,                          _,            0, _ } : unicast_untagged();
    { _,                          _,            1, 0 } : unicast_single_tagged();
    { _,                          _,            1, 1 } : unicast_double_tagged();
  }
}
```

Packet Deparsing

Deparsing

```
control MyDeparser(packet_out      packet,
                   in my_headers_t hdr)
{
   apply {
      /* Layer 2 */
      packet.emit(hdr.ethernet);
      packet.emit(hdr.vlan_tag);

      /* Layer 2.5 */
      packet.emit(hdr.mpls);

      /* Layer 3 */
        /* ARP */
      packet.emit(hdr.arp);
      packet.emit(hdr.arp_ipv4);
        /* IPv4 */
      packet.emit(hdr.ipv4);
        /* IPv6 */
      packet.emit(hdr.ipv6);

      /* Layer 4 */
      packet.emit(hdr.icmp);
      packet.emit(hdr.tcp);
      packet.emit(hdr.udp);
   }
}
```

- **Assembling the packet from headers**
- **Expressed as another control function**
  - No need for another construct
- **packet_out – defined in core.p4**
  - emit(header) – serialize the header **if** it is valid
  - emit(header_stack) – serialize the valid elements in order
- **Advantages:**
  - Decoupling of parsing and deparsing

Simplified Deparsing

```
struct my_headers_t {
    ethernet_t     ethernet;
    vlan_tag_t [2] vlan_tag;
    mpls_t     [5] mpls;
    arp_t          arp;
    arp_ipv4_t     arp_ipv4;
    ipv4_t         ipv4;
    ipv6_t         ipv6;
    icmp_t         icmp;
    tcp_t          tcp;
    udp_t          udp;
}

control MyDeparser(packet_out      packet,
                   in my_headers_t hdr)
{
    apply {
        packet.emit(hdr);
    }
}
```

- **Simply keep the header struct organized**
  - Headers will be deparsed in order

Externs

The Need for Externs

- **Most platforms contain specialized facilities**
  - They differ from vendor to vendor
  - They can't be expressed in the core language
    - Specialized computations
  - They might have control-plane accessible state or configuration
- **The language should stay the same**
  - In $P4_{14}$ almost 1/3 of all the constructs were dedicated to specialized processing
  - In $P4_{16}$ all specialized objects use the same interface
- **Objects can be used even if their implementation is hidden**
  - Through instantiation and method calling

Stateless and Stateful Objects

- **Stateless Objects: Reinitialized for each packet**
  - Variables (metadata), packet headers, packet_in, packet_out
- **Stateful Objects: Keep their state between packets**
  - Tables
  - Externs
    - $P4_{14}$: Counters, Meters, Registers, Parser Value Sets, Selectors, etc.

Object	Data Plane Interface		Control Plane Can	
	Read State	Modify/Write State	Read	Modify/Write
Table	apply()	---	Yes	Yes
Parser Value Set	get()	---	Yes	Yes
Counter	---	count()	Yes	Yes*
Meter	execute ()		Configuration Only	Configuration Only
Register	read()	write()	Yes	Yes

Counters in V1 Architecture

```
/* Definition in v1model.p4 */

enum CounterType {
    packets,
    bytes,
    packets_and_bytes
}

/* An array of counters of a given type */
extern counter {
    counter(bit<32> instance_count, CounterType type);
    void count(in bit<32> index);
}
```

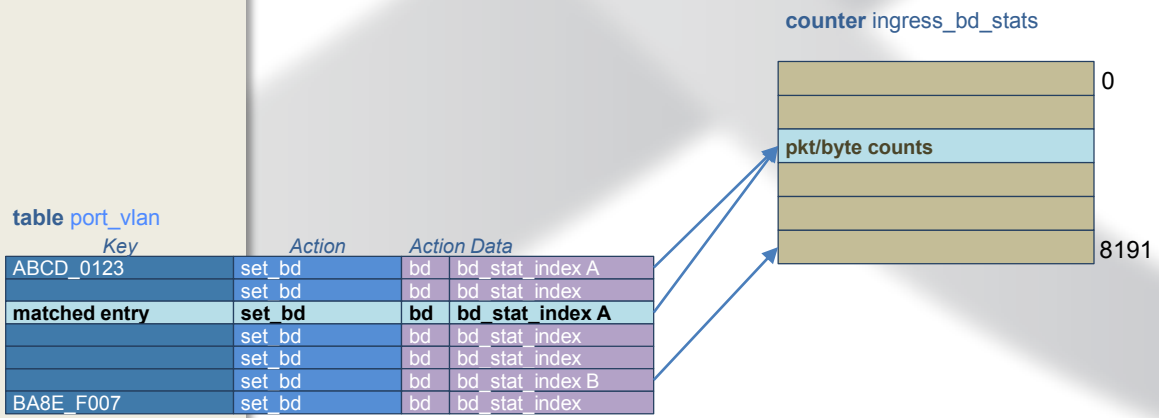
- **Extern definition contains**
  - The instantiation method
    - Has the same name as the extern
    - Is evaluated at compile-time
  - Methods to access the extern
    - Very similar to actions
    - Can return values too

- **Enums in $P4_{16}$**
  - Abstract values
    - No specific (numerical representation)

Using the V1 Architecture Counters

```
control MyIngress(inout my_headers_t       hdr,
                  inout my_metadata_t      meta,
                  inout standard_metadata_t standard_metadata)
{
  counter(8192, CounterType.packets_and_bytes) ingress_bd_stats;

  action set_bd(bit<16> bd, bit<13> bd_stat_index) {
     meta.l2.bd = bd;
     ingress_bd_stats.count((bit<32>)bd_stat_index);
  }

  table port_vlan {
     key = {
        standard_metadata.ingress_port : ternary;
        hdr.vlan_tag[0].isValid()      : ternary;
        hdr.vlan_tag[0].vid            : ternary;
     }
     actions = { set_bd; mark_for_drop; }
     default_action = mark_for_drop();
  }

  apply {
     port_vlan.apply();
     . . .
  }
}
```

- **Instantiate an extern inside the control**
  - Call the instantiation method
    - Parameters must be known at compile-time
- **Use extern's methods in actions or directly**



Meters in V1 Architecture

Definition

```
/* Definition in v1model.p4 */

enum MeterType {
    packets,
    bytes
}

extern meter {
    meter(bit<32> instance_count, MeterType type);
    void execute_meter<T>(in bit<32> index, out T result);
}
```

This is a template definition. The method will accept the parameter of any type

Color Coding:

0 – Green
1 – Yellow
2 -- Red

Usage

```
typedef bit<2> meter_color_t;

const meter_color_t METER_COLOR_GREEN  = 0;
const meter_color_t METER_COLOR_YELLOW = 1;
const meter_color_t METER_COLOR_RED    = 2;

meter(1024, MeterType.bytes) acl_meter;

action color_my_packets(bit<10> index) {
    acl_meter.execute_meter((bit<32>)index, meta.color);
}

table acl {
    key = { . . .}
    actions = { color_my_packets; . . .}
}

apply {
    acl.apply();
    if (meta.color == METER_COLOR_RED) {
        mark_to_drop();
    }
}
```

Registers in V1 Architecture

Definition

```
/* Definition in v1model.p4 */

extern register<T> {
    register(bit<32> instance_count);
    void read(out T result, in bit<32> index);
    void write(in bit<32> index, in T value);
}
```

Usage (Calculating Inter-Packet Gap)

```
register<bit<48>>(16384) last_seen;

action get_inter_packet_gap(out bit<48> interval,
                            bit<14> flow_id)
{
  bit<48> last_pkt_ts;

  /* Get the time the previous packet was seen */
  last_seen.read((bit<32>)flow_id,
                 last_pkt_ts);

  /* Calculate the time interval */
  interval = standard_metadata.ingress_global_timestamp –
             last_pkt_ts;

  /* Update the register with the new timestamp */
  last_seen.write((bit<32>)flow_id,
                  standard_metadata.ingress_global_timestamp);
}
```

Assembling the whole program

P4 Program Structure

- **P4 programs are compiled as a single module**
  - No linking of separate modules (so far)
- **P4 compiler passes the program through C Preprocessor first**
  - Use #include to modularize the code
  - P4 specification mandates only a subset of CPP features to be supported
- **Objects must be defined before they can be used**

Overall P4 Program Structure

```
/* -*- P4_16 -*- */
#include <core.p4>
#include <v1model.p4>

/********************* T Y P E S ***********************/
typedef bit<48>      mac_addr_t;
header  ethernet_t   { /* Slide 30 */ }
struct  my_headers_t { /* Slide 30 */ }

/***************** C O N S T A N T S *******************/
const mac_addr_t BROADCAST_MAC = 0xFFFFFFFFFFFF;

/************ P A R S E R S and C O N T R O L S ************/
parser  MyParser(...)         { /* Slide 38 */ }
control MyVerifyChecksum(...) { /* Slide 76 */ }
control MyIngress(...)        { /* Slide 44 */ }
control MyEgress(...)         { . . . }
control MyComputeChecksum(...) { /* Slide 76 */ }
control MyDeparser(...)       { /* Slide 65 */ }

/*************** F U L L  P A C K A G E *****************/
V1Switch(
   MyParser(), MyVerifyChecksum(),  MyIngress(),
   MyEgress(), MyComputeChecksum(), MyDeparser()
) main;
```

- **Start with Emacs-style comment to select the proper editor mode**
- **Include the core library**
- **Include the architecture-specific file(s)**
- **Define Types**
  - typedefs, headers, structs, ...
- **Define Constants**
- **Define Parsers and Controls**
- **Assemble the top-level controls in a package**
  - Package is defined by the Architecture
    - Represents the set of programmable P4 components and their interfaces
  - The name of the package must be **main**
- **That's it!**

Defining Your Own Controls

L3 Processing Function

```
control process_l3(in    my_headers_t  hdr,
                   inout l3_metadata_t l3_meta)
{
   /* Local variables for L3 processing */

   /* Tables and actions for L3 Processing */
   table ipv4_host { . . . }
   table ipv4_lpm  { . . . }
   table ipv6_lpm  { . . . }
   . . .

   apply {
      if (l3_meta.do_ipv4) {
         ipv4_host.apply();
         . . .
      } else {
         ipv6_lpm.apply();
         . . .
      }
   }
}
```

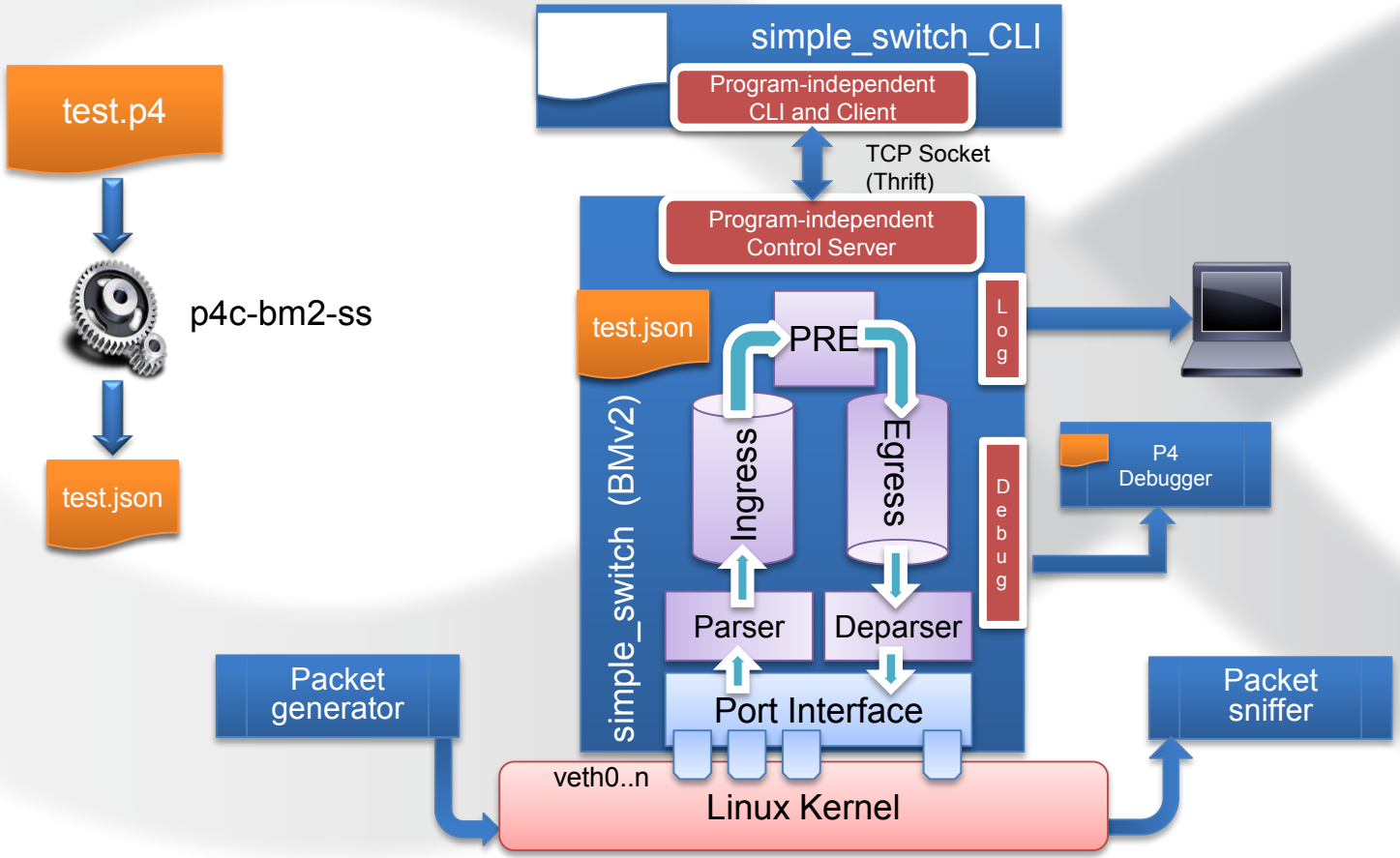
Top-Level Control

```
struct l3_metadata_t {
  bool        do_ipv4;
  nexthop_id_t nexthop;
}

control MyIngress(inout my_headers_t       hdr,
                  inout my_metadata_t      meta,
                  inout standard_metadata_t standard_metadata)
{
   l3_metadata_t  l3_meta;
   . . .

   apply {
      if (hdr.ipv4.isValid() || hdr.ipv6.isValid()) {
         l3_meta.do_ipv4 = hdr.ipv4.isValid();
         process_l3.apply(hdr, l3_meta);
      }
      . . .
   }
}
```

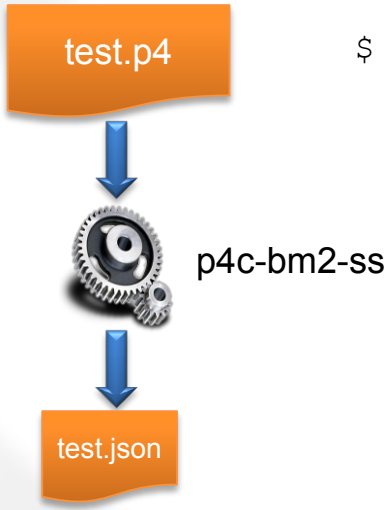
apply() method applies to controls too

P4 Toolchain

Basic Workflow

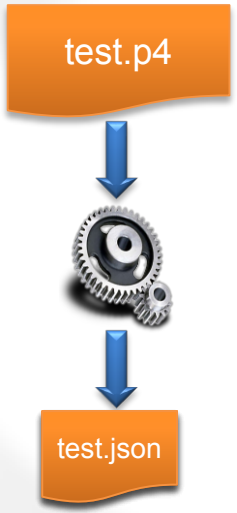


Step 1: P4 Program Compilation

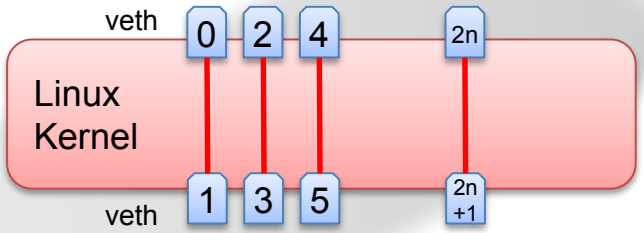



```
$ p4c-bm2-ss -o test.json test.p4
```

Step 2: Preparing veth Interfaces

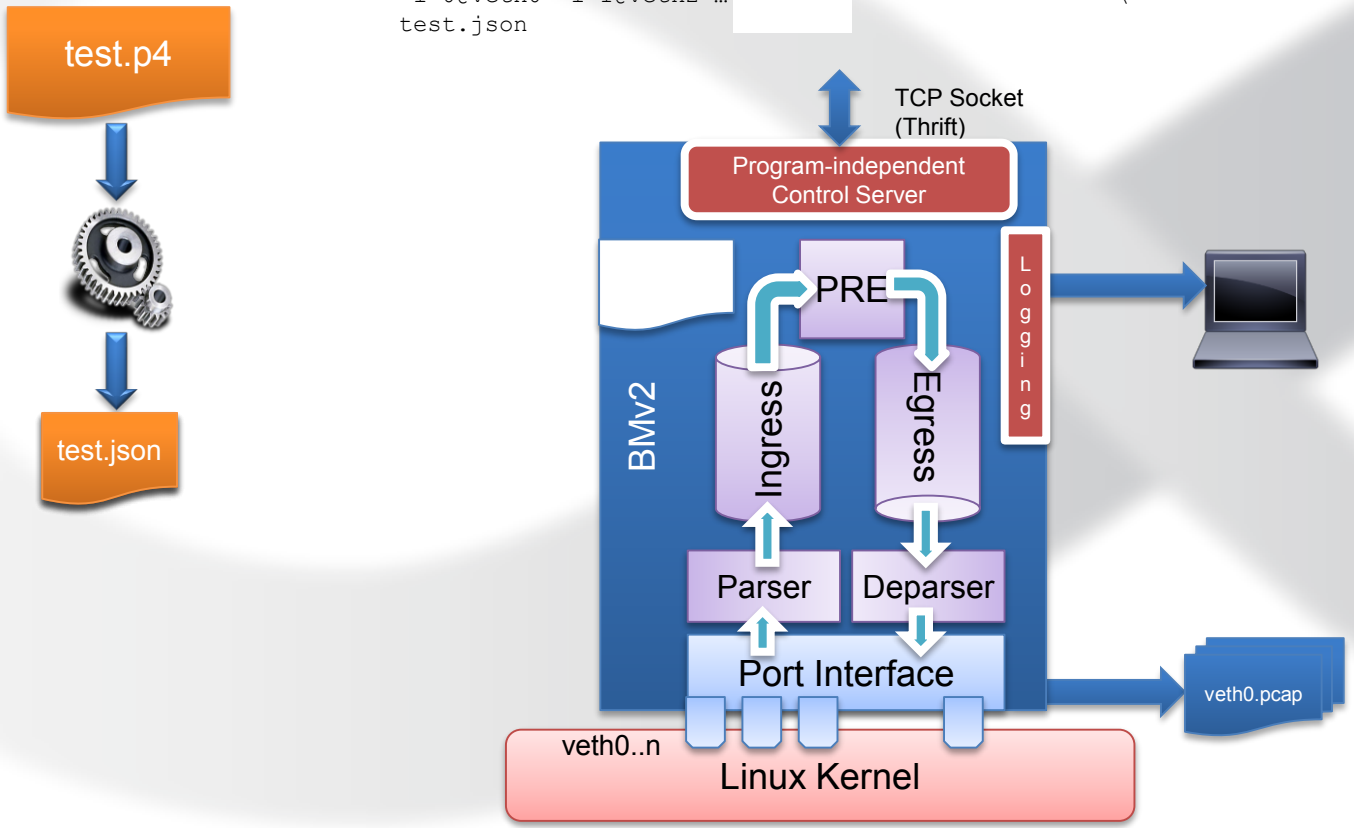



```
$ sudo ~/p4lang/tutorials/examples/veth_setup.sh

# ip link add name veth0 type veth peer name veth1
# for iface in "veth0 veth1"; do
    ip link set dev ${iface} up
    sysctl net.ipv6.conf.${iface}.disable_ipv6=1
    TOE_OPTIONS="rx tx sg tso ufo gso gro lro rxvlan txvlan rxhash"
    for TOE_OPTION in $TOE_OPTIONS; do
        /sbin/ethtool --offload $intf "$TOE_OPTION"
    done
  done
```



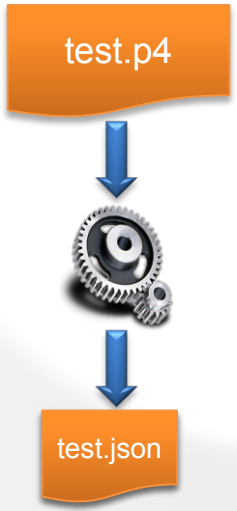
Step 3: Starting the model

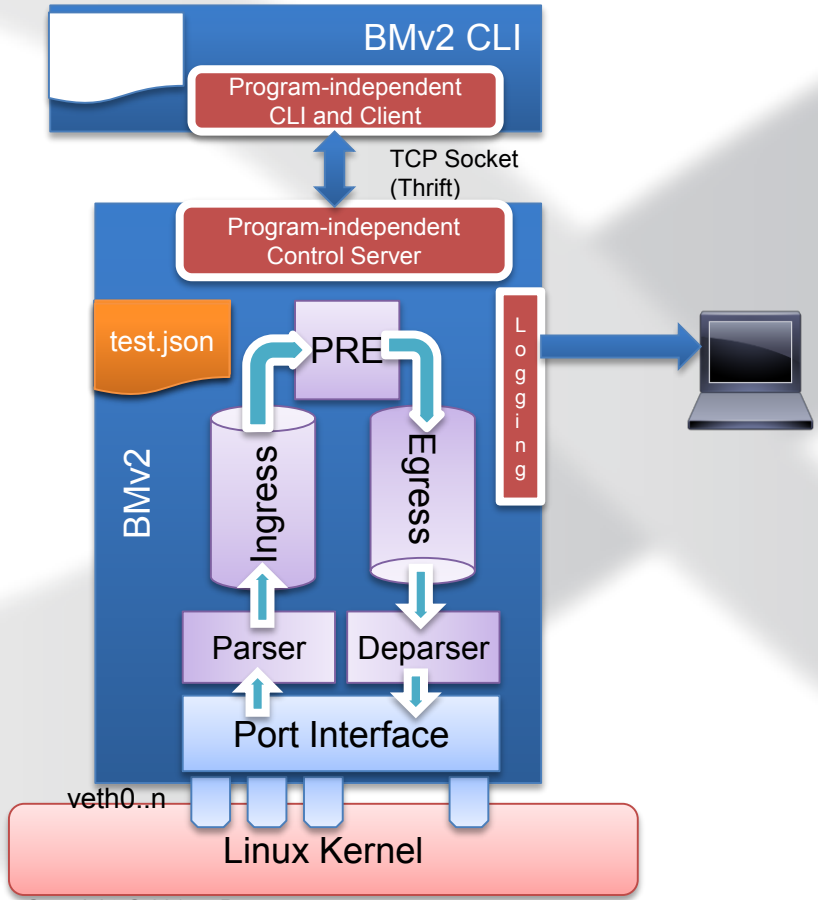
```
$ sudo simple_switch --log-console --dump-packet-data 64 \
       -i 0@veth0 -i 1@veth2 …                            \
       test.json
```



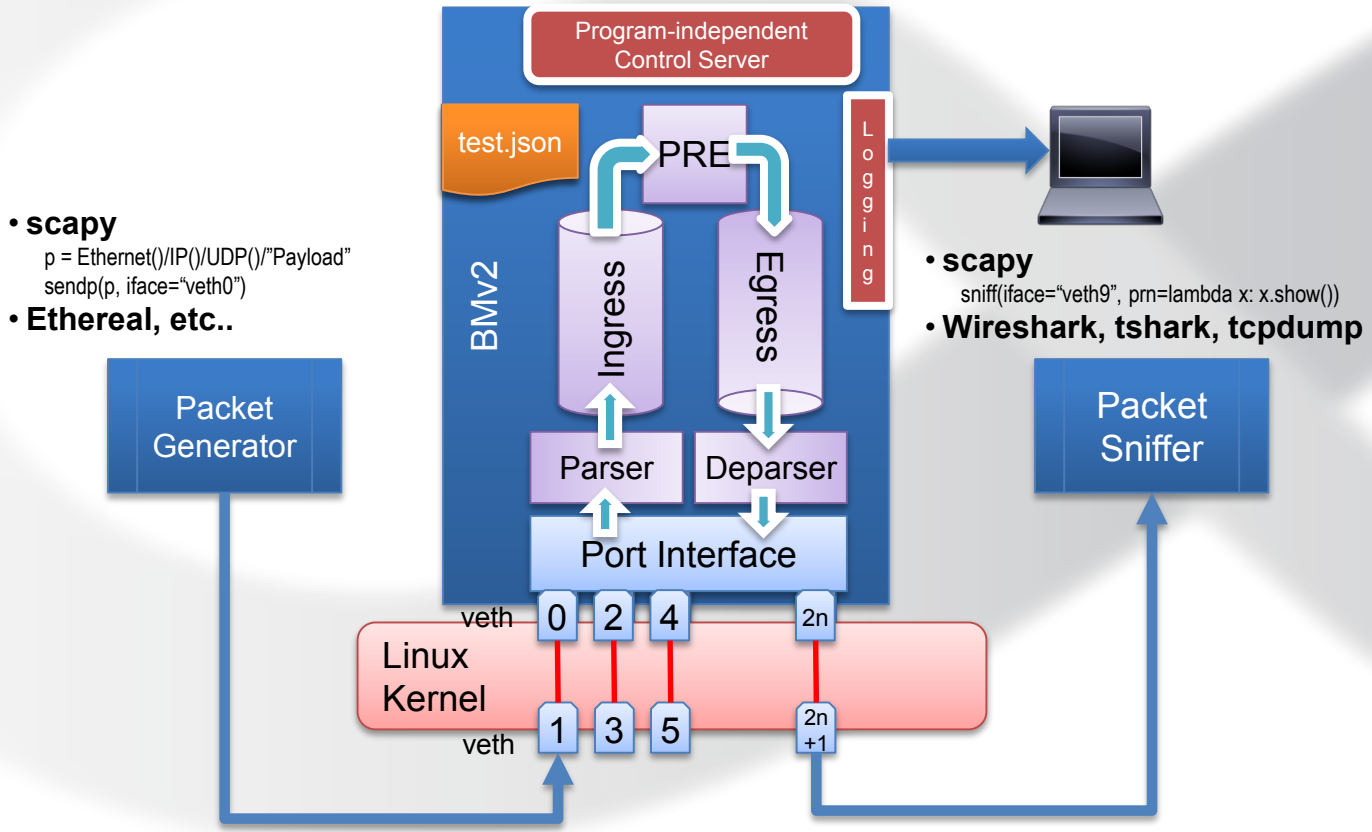
Step 4: Starting the CLI

```
$ simple_switch_CLI
```





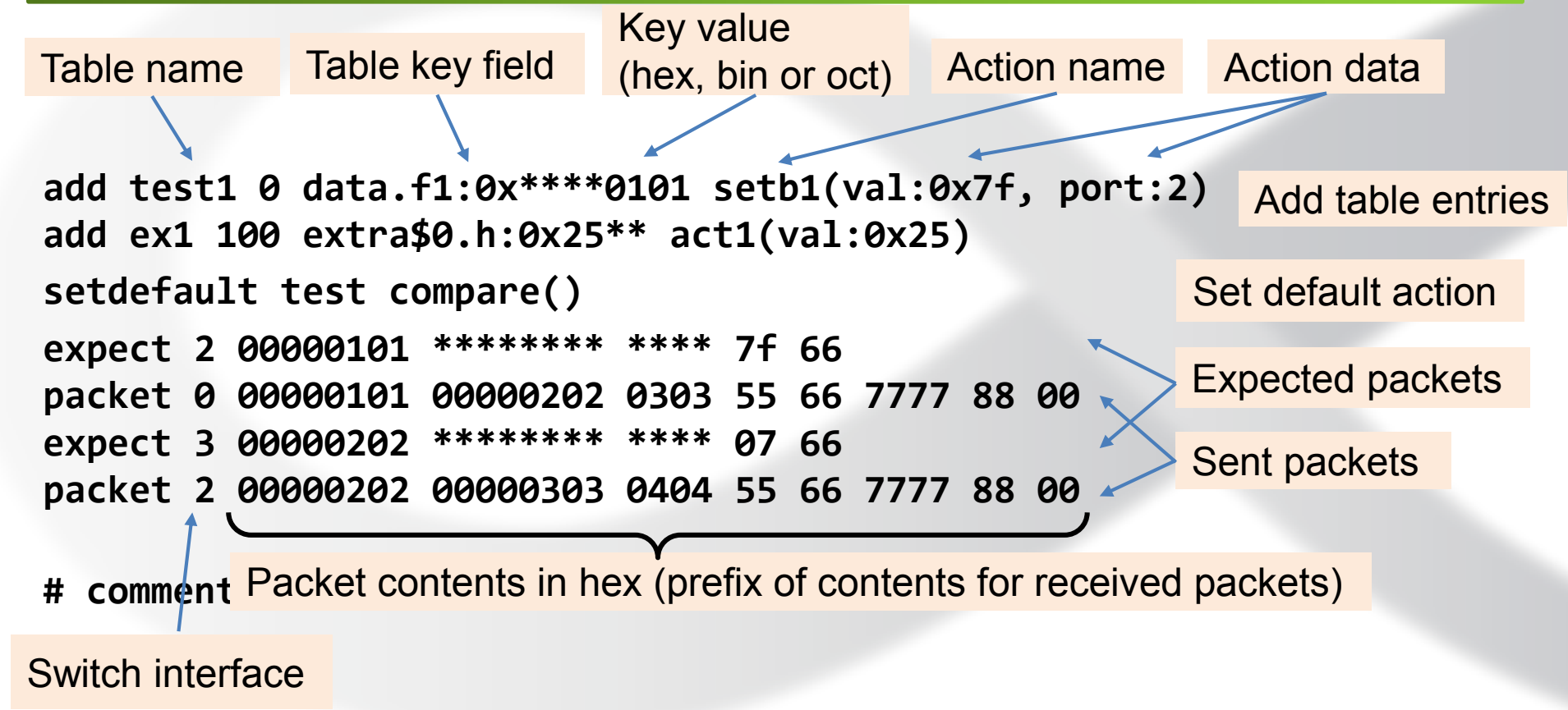
Step 5: Sending and Receiving Packets



Other tools

- **Automate small unit tests with Packet Test Framework (ptf)**

PTF (Packet Test Framework) language



Other tools

- **Mininet**
  - On a single host, emulate a **network** of multiple devices with a set of interconnecting links that you configure.
- **P4Runtime - A program independent (PI) API for**
  - Loading P4 programs into devices
  - Adding and removing table entries
  - Configuring meters and other externs, reading counter statistics
  - Either locally or remotely, over a TCP socket using Google Protocol Buffers
  - p4-api working group is writing a spec and developing the code

Extending the tools

- **All are open source - https://github.com/p4lang**
- **P4c compiler has multiple back ends already:**
  - bmv2
  - EBPF – Extended Berkeley Packet Filter, a packet filter running in Linux kernel
- **Designed to add back ends for additional devices**
  - Compiler internal documentation in **docs** directory of p4c Github repo
- **Portable Switch Architecture (PSA)**
  - p4-arch working group is developing many useful P4_16 externs
  - Good reference for how to add your own

Hardware Implementations

Tofino Chip Architecture

Tofino Ethernet Switch

Tofino Summary

World's fastest Ethernet switch – 6.5 Tbps

&

World's first Ethernet switch with a fully programmable pipeline

&

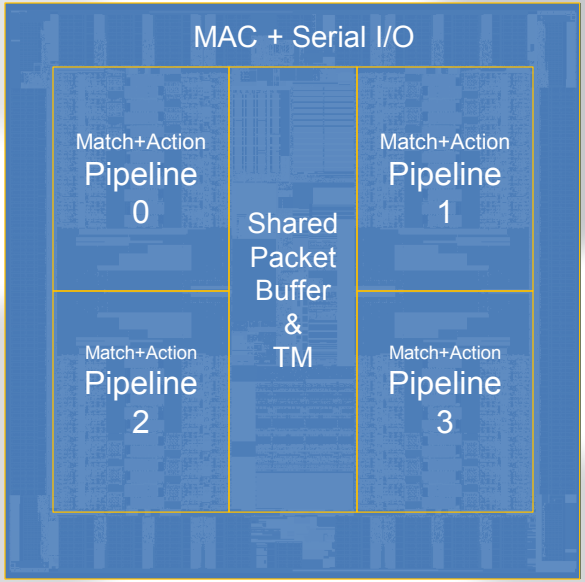
World's first Ethernet switch that is P4 programmable

&

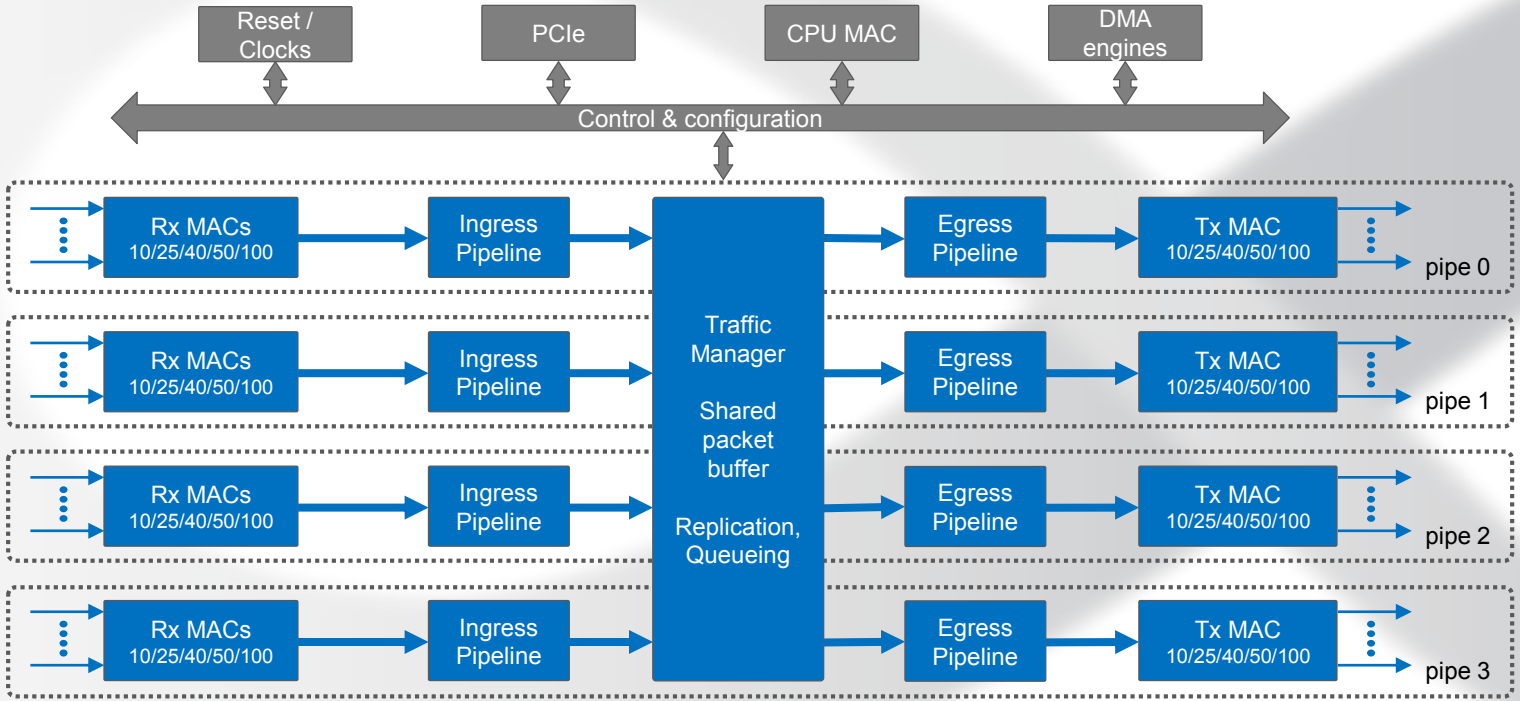
World's first Ethernet switch with no penalty on power, performance or price for programmability

6.5Tb/s Tofino™ Summary

- **State of the art design**
  - Single Shared Packet Buffer
  - TSMC 16nm FinFET+
- **Four Match+Action Pipelines**
  - Fully programmable PISA Embodiment
  - All compiled programs run at line-rate.
  - Up to 1.3 million IPv4 routes
- **Port Configurations**
  - 65 x 100GE/40GE
  - 130 x 50GE
  - 260 x 25GE/10GE
- **CPU Interfaces**
  - PCIe: Gen3 x4/x2/x1
  - Dedicated 100GE port



Tofino. Simplified Block Diagram

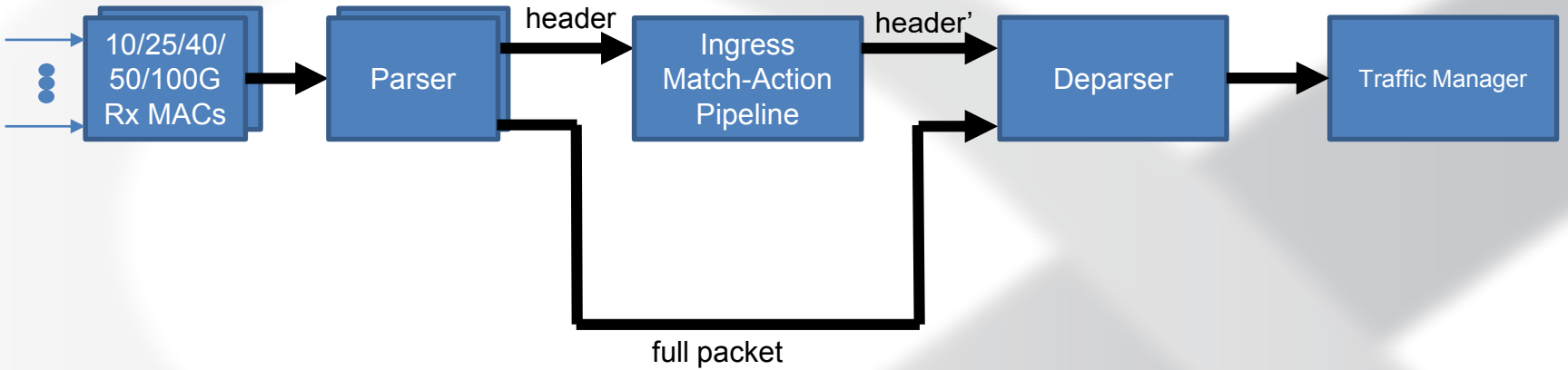


Each pipe has 16x100G MACs

+

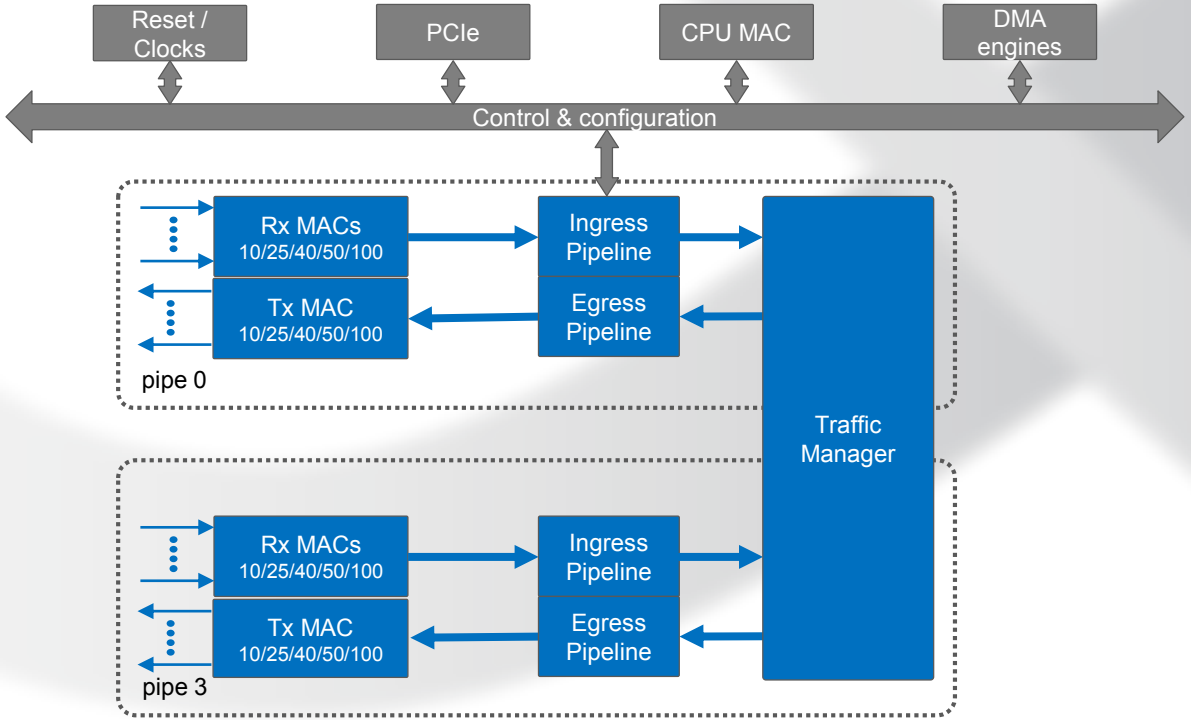
Additional MACs for recirculation, Packet Generator, CPU

The Basic Structure



Unified Pipeline

- **There is no difference between ingress and egress processing**
  - The same blocks can be efficiently shared



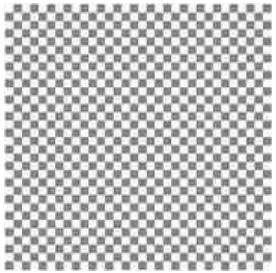
XILINX
ALL PROGRAMMABLE™

The p4c-sdnet Compiler

FPGA: the “white box” hardware chip

- **Attributes of Xilinx Ultrascale+ FPGAs in 2017:**

up to
3.7m tiles

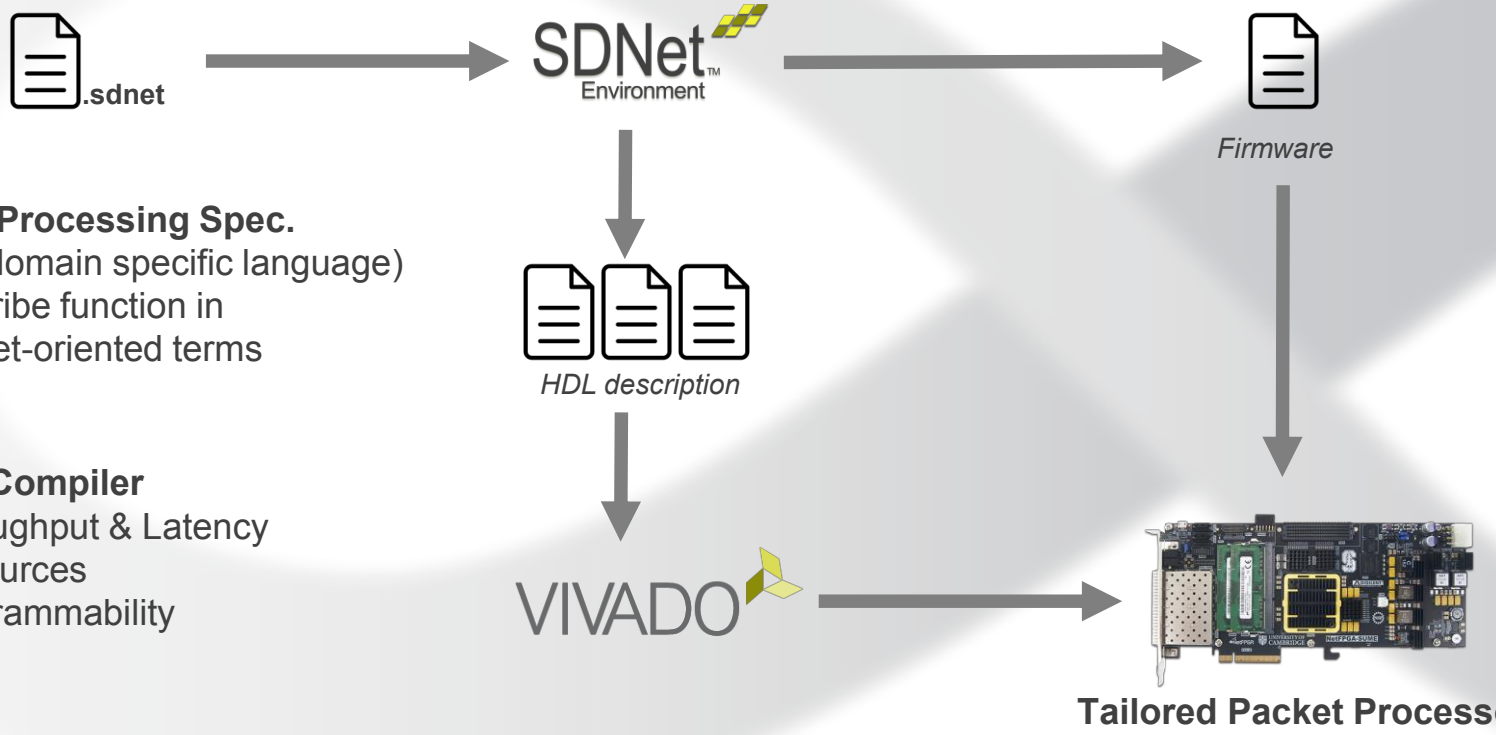


Tile: 6-input logic gate and 2 flip-flops
+ embedded function blocks & memories
+ 100G Ethernet and 150G Interlaken interfaces

Local and long-distance wiring between components
All Programmable by writing to memory

- **Can now implement complex packet processing on single chip**
- **Beyond single chips**
  - Multiple FPGAs (e.g. Corsa SDN data planes)
  - FPGA fast paths, CPU slow paths & control
  - FPGA smart paths, ASIC dumb switches

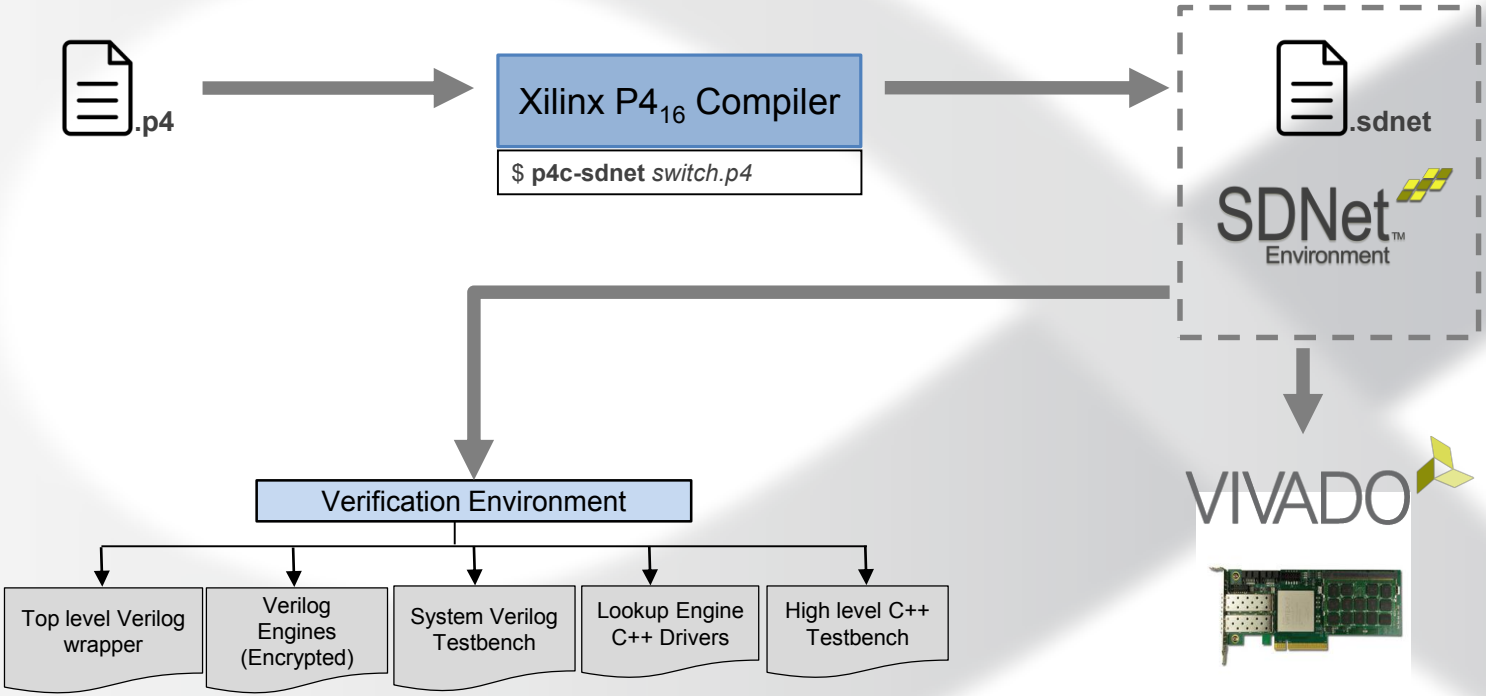
Xilinx SDNet Design Flow & Use Model



Packet Processing Spec.
- PX (domain specific language)
- describe function in packet-oriented terms

SDNet Compiler
- Throughput & Latency
- Resources
- Programmability

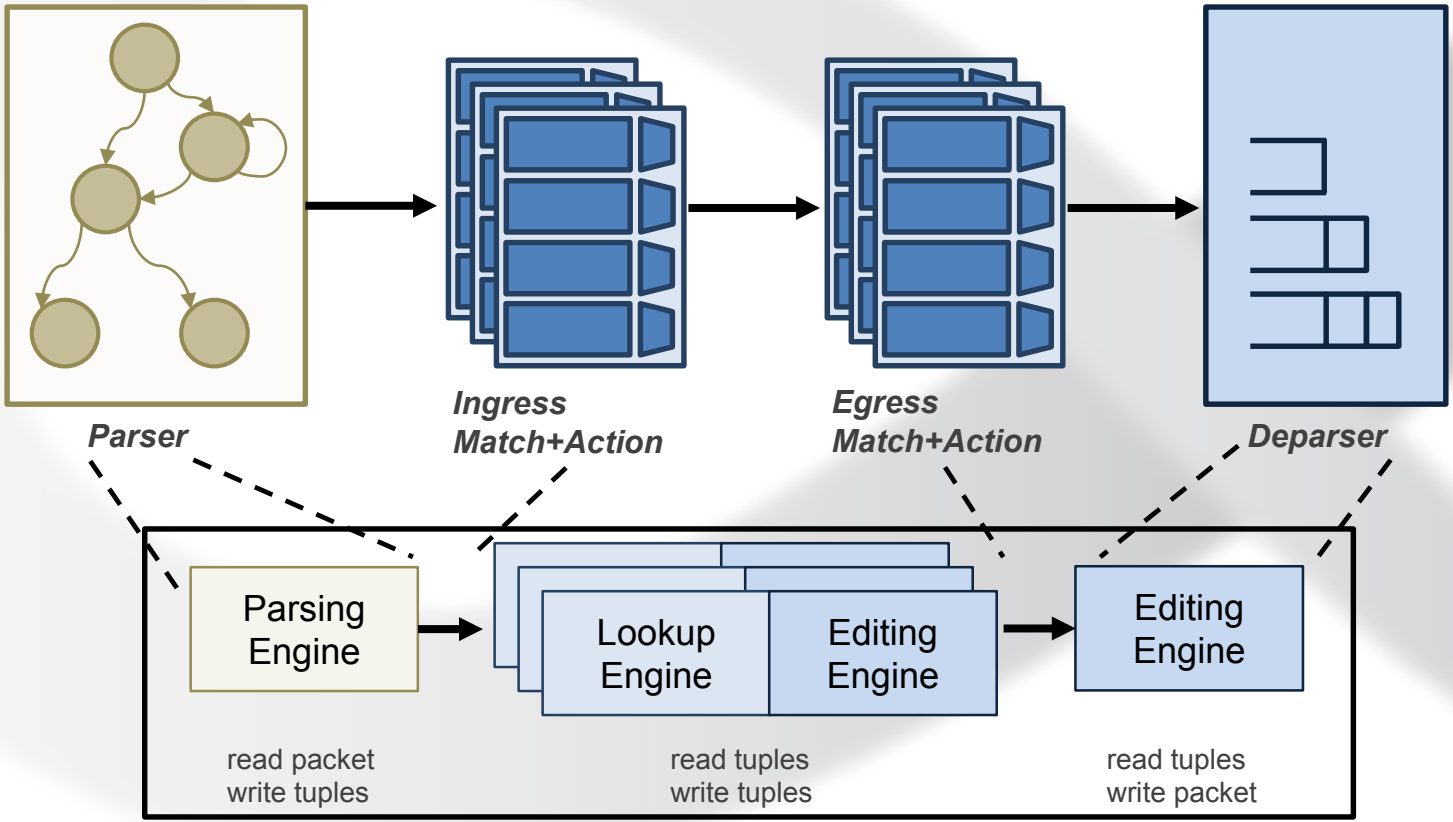
Xilinx P4 Design Flow & Use Model



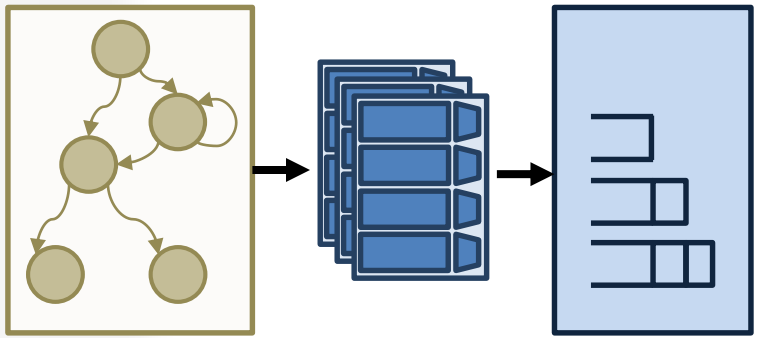
Considerations When Mapping to SDNet

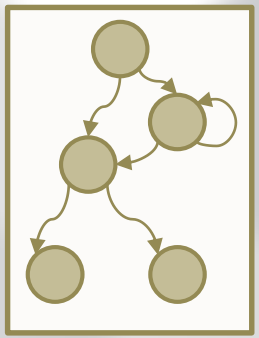
- **Identifying parallelism within P4 parser and control blocks**
  - table lookups
  - actions
  - etc.
- **P4 packet processing model**
  - extract entire header from packet
  - updates apply directly to header
  - deparser re-inserts header back into packet
- **SDNet packet processing model**
  - stream packet through “engines”
  - modify header values in-line without removing and re-inserting

Mapping P4 Architectures to SDNet



Support for Multiple Architectures



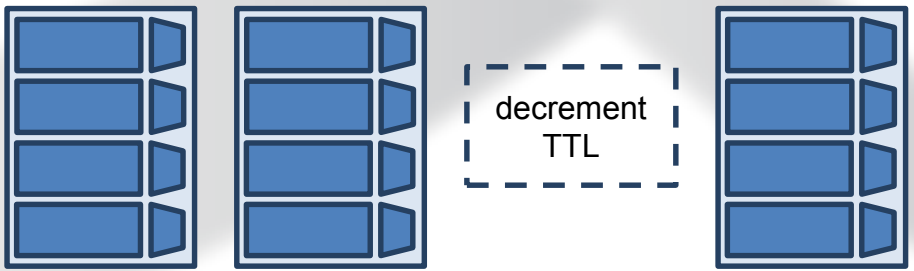


- Single Match+Action Pipeline
  - simple updates to packet headers
- Only Parser
  - pull information from packet w/o updates

Providing Externs using Custom User Logic

```
header ipv4_t { … }

extern void decrement_ttl (inout ipv4_t ip);

Control MyIngress (
   inout my_headers_t          hdr,
   inout my_metadata_t         meta,
   inout standard_metadata_t std_meta)
{

   ...

   apply
   {
      ...

      if (hdr.ip.isValid())
         decrement_ttl(hdr.ip);

      ...

   }
}
```

- **User writes a custom Verilog component**
  - decrement_ttl.v
- **use P4's extern construct in the switch description file**
- **p4c-sdnet flow generates hook so module can be added into the bitstream**

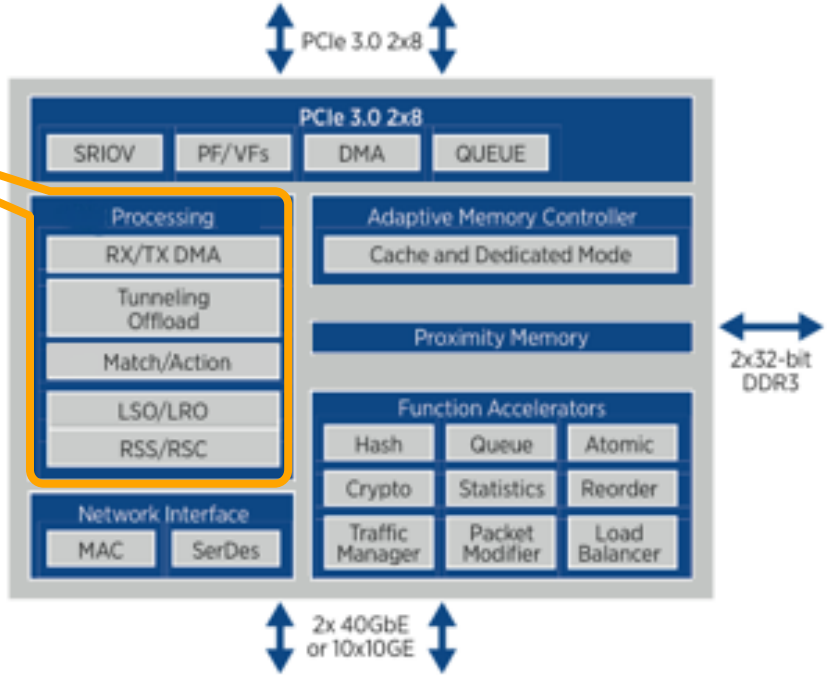


Running P4 on Network Flow Processors and Agilio® SmartNICs

Network Flow Processor (NFP) Silicon Family

- **Growing family of software compatible devices: C programmable, run to completion**
  - 16 to 120+ cores @ up to 1.4 GHz
  - 8 threads per core => up to ~1000 threads
- **Flexible network media: 10M to 100G Ethernet, Interlaken to support other media**
- **Multiple PCIe buses: up to 4 buses - each 8 lane PCIe gen 3, root or endpoint**
- **Throughput: 20 to 200+ Gbit/s (>180 Mpps)**
- **Flexible memory: up to 24GB DRAM - for lookup and state tables, meters, packet buffers => millions of entries**
- **On-chip accelerators: classification, state tables, queues, metering, statistics, ordering/ reassembly, encryption, QoS…**

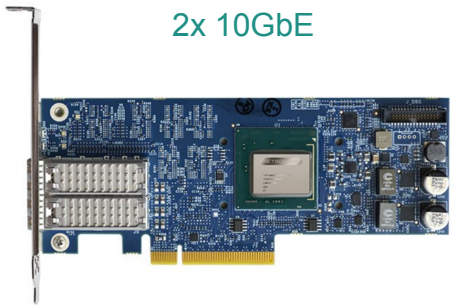
Example: NFP-4xxx



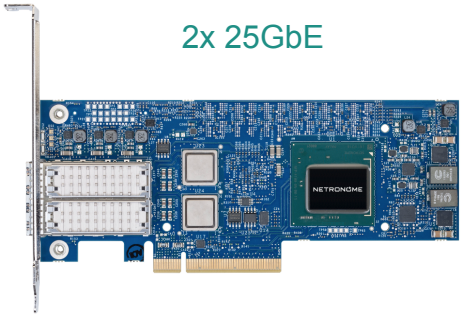
Agilio CX SmartNIC Family

- **Competitively priced NIC with loadable firmware**
  - Custom programmed (P4 / eBPF*) or pre-programmed (Core NIC, OVS, Contrail vRouter)
- **Optimized for standard server based cloud data centers**
- **Based on Netronome's Network Flow Processor 4xxx chips**
- **Equipped with 2GB DRAM**
- **Low Profile Half Length PCIe form factor, <25W Power**

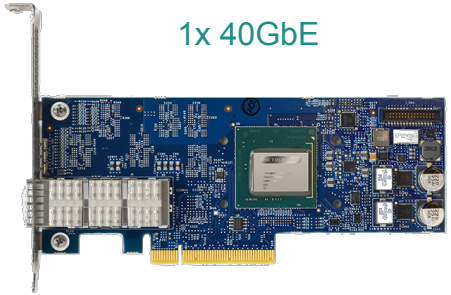
2x 10GbE



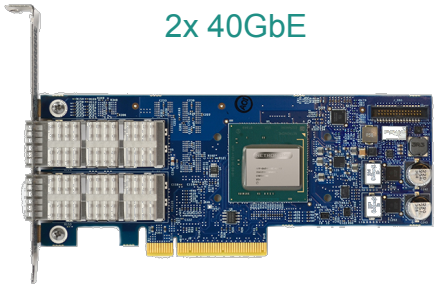
2x 25GbE



1x 40GbE



2x 40GbE



Also available: Agilio LX with 8GB DRAM, 2x40G or 1x100G

* *eBPF support is in development*

P4 Integrated Development Environment

Programmer Studio IDE

Edit - Build - Run

Debug with Code / Data Visualization

=

Parse - Match - Act Dataplane Program

app.P4

Compile

Intermediate Representation

app.IR

Extension = "extern"

plugin.C

Compile / Link

rules. JSON

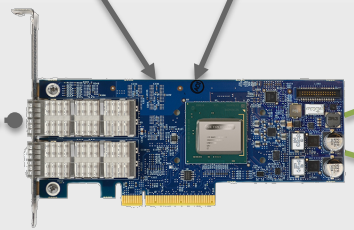
Run-time I/F

app.firmware

Run-time interaction
Alternative: use API

Native code for SmartNIC

10G / 25G / 40G / 100G Network



SR-IOV netdev: vf0

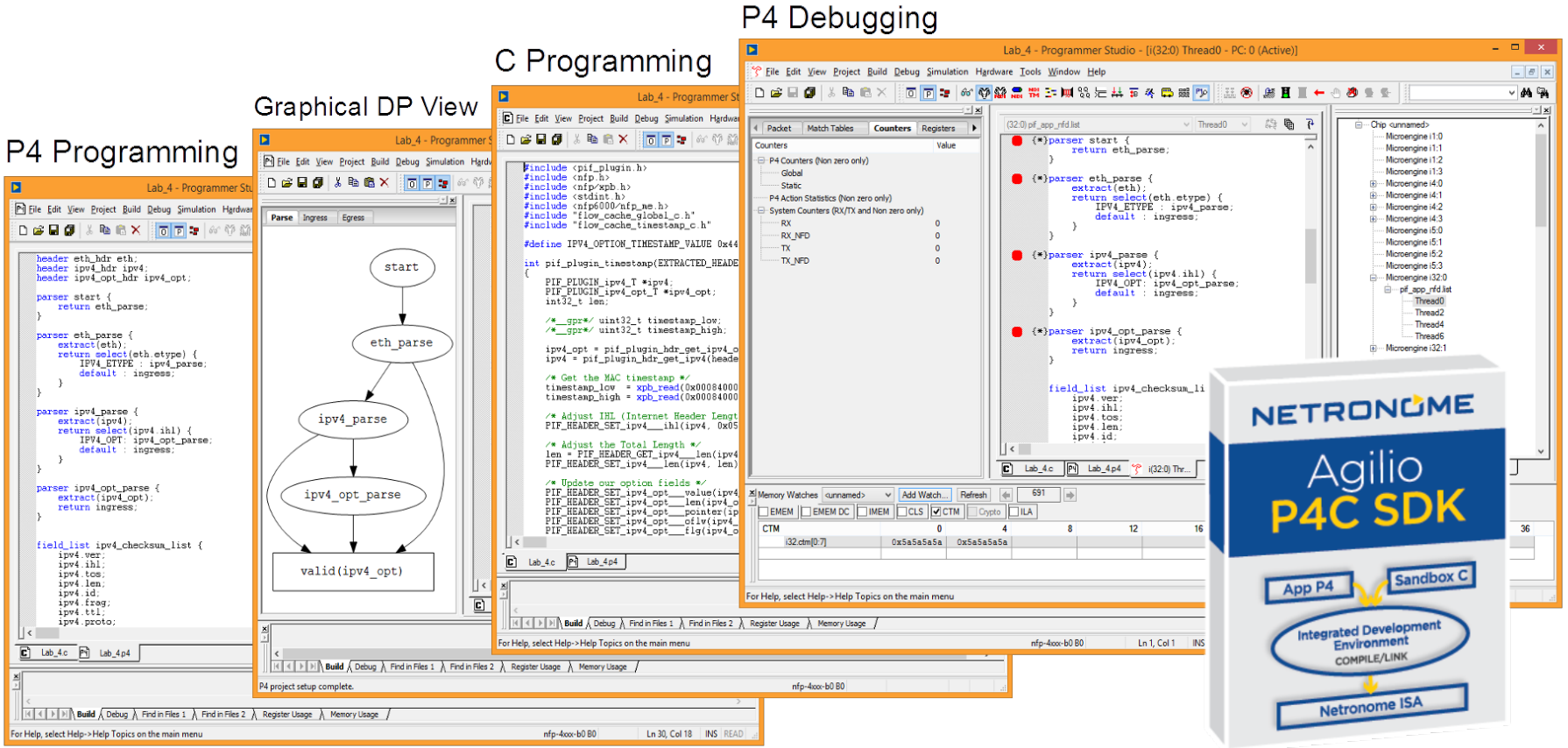
SR-IOV DPDK: vf1

VNF or VM or App

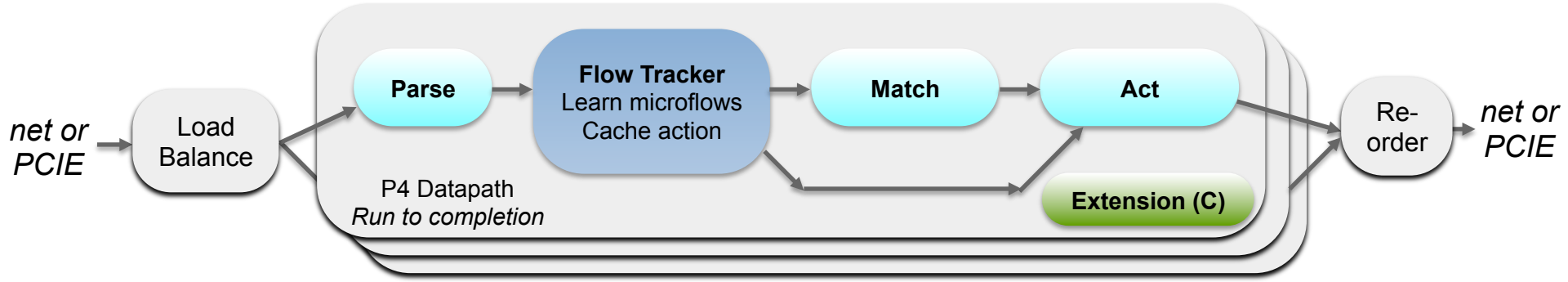
Linux Server

Alternate flow: compile P4 to eBPF, transparently accelerate eBPF (JIT to NFP)

IDE - Visualization Examples



Datapath Software Architecture



Pool of worker threads on flow processing cores

- **Load balancer distributes each packet to next available thread for optimum throughput**
- **Hardware assisted reordering ensures packet order is maintained**
- **Matching performed using DRAM-backed "algorithmic TCAM"**
- **Actions: forward (to Ethernet or PCIe), clone, edit packet, QoS (e.g. metering), counters…**

=> Conveniently supports varying runtime per packet + high throughput / flow capacity

Next Steps

- **Use Agilio™ SmartNICs with pre-programmed dataplane software**
  - Core NIC
  - OVS (with / without Conntrack)
  - Contrail vRouter
- **Program Agilio SmartNICs**
  - Using P4, C, eBPF/XDP* …
- **Participate in open source and standards evolution**
  - p4.org, open-nfp.org, openstack.org, openvswitch.org, opencontrail.org, iovisor.org, opensourcesdn.org, opnfv.org, linuxfoundation.org …
  - Example: VNF offload API (@ OPNFV + other bodies)

** eBPF support is in development*

Next Steps

- What are researchers doing with P4?
- How to get involved?
- Where to get the code?

P4 Research

P4 – NetFPGA

Stephen Ibanez, et al.

Using P4 to extend the reach of NetFPGA beyond the HW community, and using FPGAs to amplify the power of P4.

PISCES

Muhammad Shahbaz, et al.

Introducing a P4 datapath into the Open vSwitch software-based switch.

NetPaxos

Huynh Tu Dang, et al.

Consensus at network speed.

NetCache

Xin Jin, et al.

Fast, in-network caching for key-value stores.

Executable Formal Semantics of P4

Ali Kheradmand, et al.

Expressing P4 semantics in the K Framework.

A Program Logic for P4

Foster, et al.

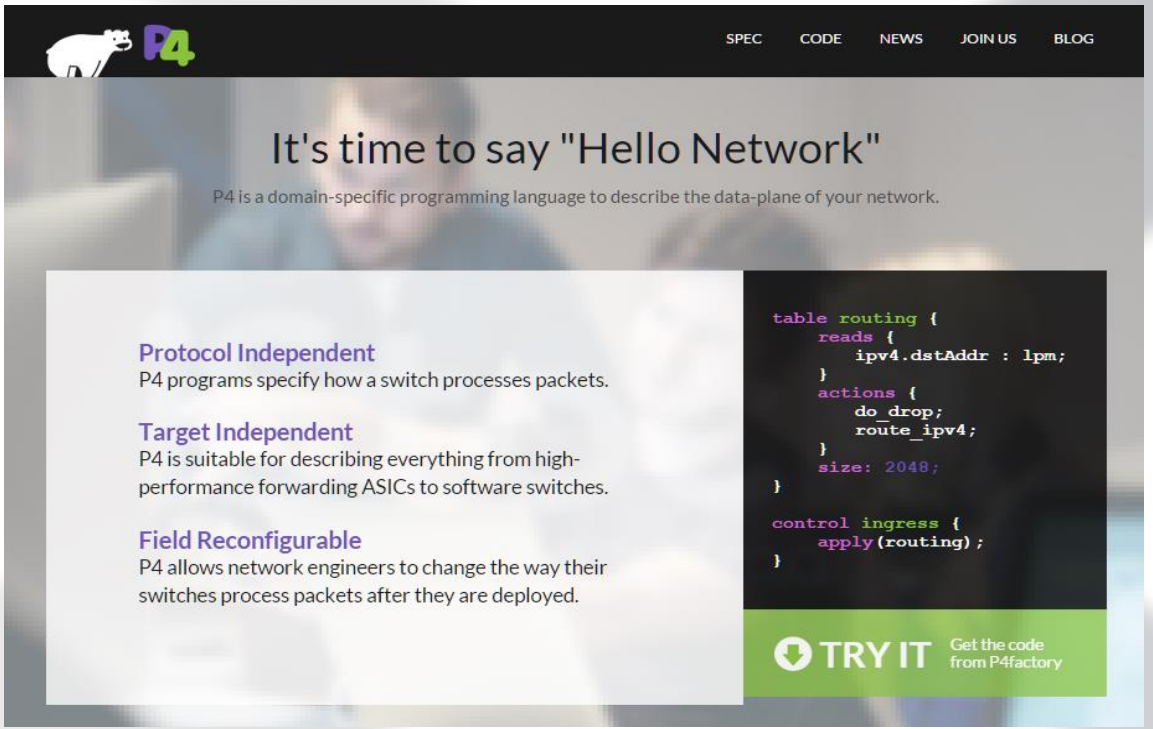
Automated P4 verification by reduction to SMT.

Future Research Topics

- **Supporting operations on payloads at Tbits/sec**
  - Deep packet inspection, encryption, compression
- **Virtualized instances**
  - Enable different tenants to program their forwarding differently
- **High performance stateful features**
  - Locking, or some other new approach

The P4 Language Consortium

- http://p4.org
- Consortium of academic and industry members
- Open source, evolving, domain-specific language
- Permissive Apache license, code on GitHub today
- Membership is free: contributions are welcome
- Independent, set up as a California nonprofit



P4.org Membership

Original P4 Paper Authors:

Barefoot Networks, Google, Intel, Microsoft, Princeton University, Stanford University

Operators/ End Users

Alibaba Group, Baidu 百度, FOX, Goldman Sachs, at&t, Comcast, Google, kt, Microsoft, Tencent 腾讯, SK telecom

Systems

Arista, Brocade, Cisco, DAHO Networks, Corsa, Dell, Hewlett Packard Enterprise, Huawei, Inventec, Intracom Telecom, Juniper Networks, mantis, Netberg, NetBRIC, NoviFlow, OpenxNet, ZTE, Xentech

Targets

Aeponyx, Atomic Rules, Barefoot Networks, Cavium, EZchip, Intel, Netcope Technologies, Mellanox Technologies, MoSys, Broadcom, centec networks, freescale, Innovium, Netronome, PLUMgrid, vmware, Marvell, Stordis, Xilinx

Solutions/ Services

EstiNet, happiest minds, Global Hitech, SDNLAB 专注网络创新技术, FLOW Research

Academia/ Research

Bii 天地互连, Cornell University, National Chiao Tung University, Korea University, Princeton University, salzburgresearch, uni.lu Université du Luxembourg, Università della Svizzera italiana, Eötvös Loránd University, Politecnico Milano 1863, Polytechnique Montréal, Stanford University, UFRGS Universidade Federal do Rio Grande do Sul, VirginiaTech Invent the Future

- **Open source**, evolving, domain-specific language
- Permissive Apache license, code on GitHub today
- **Membership is free**: contributions are welcome
- Independent, set up as a California nonprofit

Acknowledgements

- Antonin Bas
- Gordon Brebner
- Mihai Budiu
- Calin Cascaval
- Chris Dodd
- Nate Foster
- Vladimir Gurevich
- Andy Keep
- Changhoon Kim
- Nick McKeown
- Edwin Peer
- Ben Pfaff
- Cole Schlesinger
- Lorenzo Vicisano
- Han Wang

Thank you